Mykonos Security User Guide

Mykonos Security User Guide

Copyright © 2011 Mykonos Software Inc.

Table of Contents

1. Introduction 1
    1. Overview 1
    2. Major components 2
    3. Features 2
    4. Support 3
2. Deployment Overview 4
    1. Mykonos Web Security network placement 4
        1.1. Proxy-like placement 4
    2. Options for providing a load balanced environment 5
    3. SSL traffic consideration 5
    4. Limitations 6
3. Installing Mykonos Web Security 7
    1. Configuring MWS for the First Time 7
    2. MWS Setup 7
        2.1. Network Configuration 8
        2.2. SSL Certificates 8
        2.3. Authentication Method 9
4. Configuring Mykonos Web Security 10
    1. Basic Configuration 10
    2. Global Configuration 10
        2.1. Configuring Backups 11
        2.2. Configuring Alerts 12
    3. Processor Configuration 12
    4. Creating an Application Profile 13
        4.1. Adding Pages 13
        4.2. Configuring the URL Pattern 14
        4.3. Configuring the Server Health Checks 14
        4.4. Configuring the Target Servers 15
        4.5. Creating Page Specific Configurations 15
    5. Configuring SSL 15
        5.1. Enabling SSL to the Client 16
        5.2. Enabling SSL to the backend 16
    6. Configuring Mail Server 16
5. Managing the Appliance 17
    1. Restart/Shutdown 17
    2. System Updates 17
    3. Licensing 17
    4. Troubleshooting and Maintenance 17
        4.1. Managing the Services 17
        4.2. Viewing Logs 17
    5. Backup and Recovery 18
        5.1. Restoring System Configuration from Backup 18
        5.2. Backing up and Restoring Console Data 18
6. Processor Reference 19
    1. Complexity Definitions 19
    2. Security Processors 20
    3. Tar Trap Processors 20
        3.1. Access Policy Processor 20
        3.2. Ajax Processor 23
        3.3. Basic Authentication Processor 25
        3.4. Cookie Processor 30

3.5. File Processor 32
3.6. Hidden Input Processor 34
3.7. Hidden Link Processor 37
3.8. Query String Processor 38
3.9. Robots Processor 39
4. Activity Processors 42
4.1. Custom Authentication Processor 42
4.2. Cookie Protection Processor 46
4.3. Error Processor 47
4.4. Header Processor 50
4.5. Method Processor 55
4.6. User Agent Processor 57
5. Tracking Processors 59
5.1. Etag Beacon Processor 59
5.2. Client Beacon Processor 61
6. Response Processors 64
6.1. Block Processor 64
6.2. Request Captcha Processor 64
6.3. CSRF Processor 81
6.4. Header Injection Processor 85
6.5. Force Logout Processor 85
6.6. Strip Inputs Processor 86
6.7. Slow Connection Processor 87
6.8. Warning Processor 87
7. Reporting 90
1. Generate Report 90
1.1. Report Schedules 90
1.2. Scheduling a Report 90
1.3. Report History 90
8. Using the Security Monitor 91
1. Navigation Menu 91
2. Dashboard 91
3. Incidents, Sessions, and Profiles Charts 91
4. Incidents Menu Filter Options 91
5. Complexity Definitions 92
6. Incidents List Widget 92
6.1. Incidents List Details 92
7. Incident Widget 93
7.1. Incidents Widget Details 93
8. Sessions Widget 94
9. Profile List Widget 94
9.1. Profiles List Details 94
10. Profile Widget 94
10.1. Editing a Profile 95
10.2. Configuring Responses 95
11. Location Widget 96
12. Environment Details 96
9. Auto-Response Configuration 98
1. Using the Editor 101
10. Command Line Interface 102
1. Supported arguments 102
1.1. Data Sources 102
1.2. Formatting 103
1.3. Example Report 103

A. Incident Methods 104

List of Tables

1.1. 1
4.1. 11
4.2. 12
4.3. 14
4.4. 15
6.1. 21
6.2. 23
6.3. 25
6.4. 31
6.5. 32
6.6. 34
6.7. 37
6.8. 38
6.9. 40
6.10. 42
6.11. 46
6.12. 48
6.13. 51
6.14. 56
6.15. 58
6.16. 59
6.17. 61
6.18. 64
6.19. 64
6.20. 82
6.21. 85
6.22. 86
6.23. 86
6.24. 87
6.25. 87
8.1. 92
8.2. 93
8.3. 94
8.4. 95
8.5. 95
9.1. 98
9.2. 101
A.1. 104

Chapter 1. Introduction

1. Overview

Summary

Mykonos Web Security (MWS) is a smart way to secure websites and web applications against hackers and attacks leading to possible fraud and or theft. Using it's Web Intrusion Prevention System (WIPS), MWS employs deception methods to detect, track, profile and prevent hackers all in real-time. Unlike signature-based approaches, MWS is the first technology that injects detection points to proactively identify attackers before they do damage; all without any false positives. Mykonos goes beyond the layer 3 IP address to not only track the individual attacker, but profile their behavior and deploy counter measures as well. With the MWS software, administrators are liberated from writing rules, analyzing massive log files or monitoring other web security consoles. MWS neutralizes threats as they occur preventing the loss of data and potentially saving companies millions of dollars from fraud or lost revenue.

What problem does it solve?

MWS is a software product that protects websites and web applications from hackers, fraud and theft.

Why is it unique as a security product?

MWS software uses deceptive techniques that make hacking a MWS protected website more time consuming, tedious and above all, costly. The software injects detection points into the hosted web application code at the time it is served by the web server. MWS will analyze the detection points for malicious activites. It then removes these detection points out on the way back into the server. The detection points have no damaging or intrusive effect on your website and are invisible to normal users. The result is MWS will work with any web entity with no need to alter a single line of code within the website or web application. All the detection points are inserted by the Mykonos Web Security software, thus reducing the chance of false positives to near zero.

What does the software do?

There are four main features of Mykonos Web Security.

Table 1.1.

Step 1 **Detect**		MWS uses detection points called tar traps to detect threats and attackers without generating false positives. These tar traps flag when hackers are mapping out your site and doing reconnaissance. They are scanning for vulnerabilities to exploit against your applications.
Step 2 **Track**		MWS uses methods including persistent tokens and fingerprinting to track not only IPs but browsers, software and scripts.

Step 3 **Profile**		Mykonos Web Security creates a smart profile for each user comprised of the number of threats, threat level, and activity. This profile will help you understand not only the threats against your applications but will help you determine the best way to respond.
Step 4 **Respond**		Upon detecting, tracking and profiling the threat, MWS provides you with many manual, automated and customizable block, warn and deceive options.

What traffic does it inspect?

The Mykonos Web Security software inspects only HTTP (TCP/IP port 80) and HTTPS (TCP/IP port 443) traffic, functioning as a reverse proxy.

How is it managed?

The MWS logs all incidents to a database creating attacker profiles. This allows security administrators to monitor activity real-time through a web-based monitoring console. Administrators can respond manually or by applying automated hacking prevention policies.

2. Major components

Major Mykonos Web Security components include but are not limited to:

- The HTTP/HTTPS gateway proxy for acceptance and translation of raw traffic and interaction with the security engine. This is imparative to having the appliance serve as an inline proxy. This layer is also responsible for SSL encryption/decryption (if needed) and compression.
- The security engine and profile database are core components for managing all aspects of real-time security.
- The response rules engine automates responses to malicious behavior identified by the security engine, all occurring in real-time.
- A web monitoring console that provides real-time hacker monitoring, MSW application configuration, reporting and report generation.
- MWS processors are objects that contain specific security exploits and associated logic for the security engine to detect, interpret, and differentiate as malicious behavior from expected behavior.

3. Features

Abuse detection processors	A library of HTTP processors that inject specific abuse tar traps into a web application's code stream. Tar traps identify abusive users who are trying to establish attack vectors such as SQL injection, cross-site scripting, session hi-jacking and cross-site request forgery.

HTTP capture	Software that captures, logs, and displays HTTP traffic to analyze for security incidents.
Abuse responses	An established set of repsonses that enables administrators to act on application abuse with session-specific warnings, blocks, and additional checks.
Abuse profiles	MWS maintains a historical profile database of known application abusers and all of their historical malicious activity against the MWS protected application, for analysis and sharing.
Tagging and re-identification	This feature enables allows the MWS software to re-identify abusive users and apply persistent responses over time and across sessions.
Policy expressions	Simple regular expression syntax is allowed for writing automated, application-wide countermeasures for the MWS software's policy engine.
Email alerts	The MWS software can send email alerts when specific incidents or incident patterns occur.
Monitoring console	MWS has a web-based monitoring and analysis interface allowing administrators to drill down into application sessions, security incidents, and abuse profiles. Administrators can also manage and monitor manual and automated responses from this interface.
Web-based configuration	This is a browser-based configuration interface for all application configuration and deployment options. See monitoring console.
Software and hardware delivery support	The MWS software can be distributed as VMware virtual machine (VM) image, an Amazon Web Serivces (AWS) cloud virtual machine, or drop-shipped as a pre-built hardware appliance on HP hardware.
SSL inspection	The MWS software can process SSL traffic via passive decryption or certificate termination.
Multi-application protection	MWS allows single application processes and can secure traffic for multiple application domains. For example, the MWS can be used to secure www.mykonossoftware.com as well as www.someotherwebsite.com.

4. Support

For any issues or questions regarding this document or the Mykonos Web Security software, please contact Mykonos customer support during regular business hours, 0800 - 1800 EST:

- support@mykonossoftware.com
- 1-877-88-WINGS

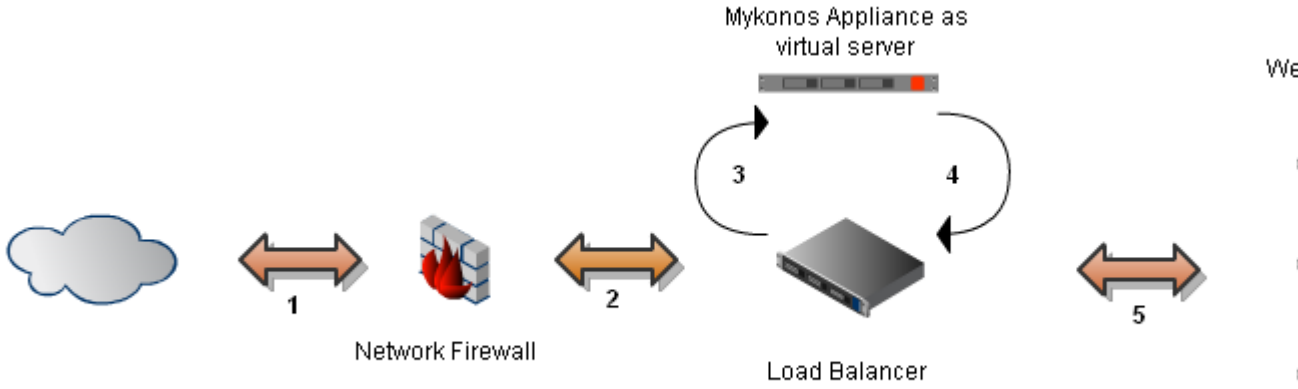
Chapter 2. Deployment Overview

In the context of a network diagram, Mykonos Web Security should be considered a reverse web proxy server. Because MWS functions at layer 7 in the OSI model, it only accepts HTTP/HTTPS (TCP ports 80 and 443) connections. *The 100Mbps units have 2 network interfaces, and the 600Mbps units have 6. All interfaces can be configured for both inbound or outbound traffic as long as the source/destinations have dedicated IPs.*

1. Mykonos Web Security network placement

1.1. Proxy-like placement

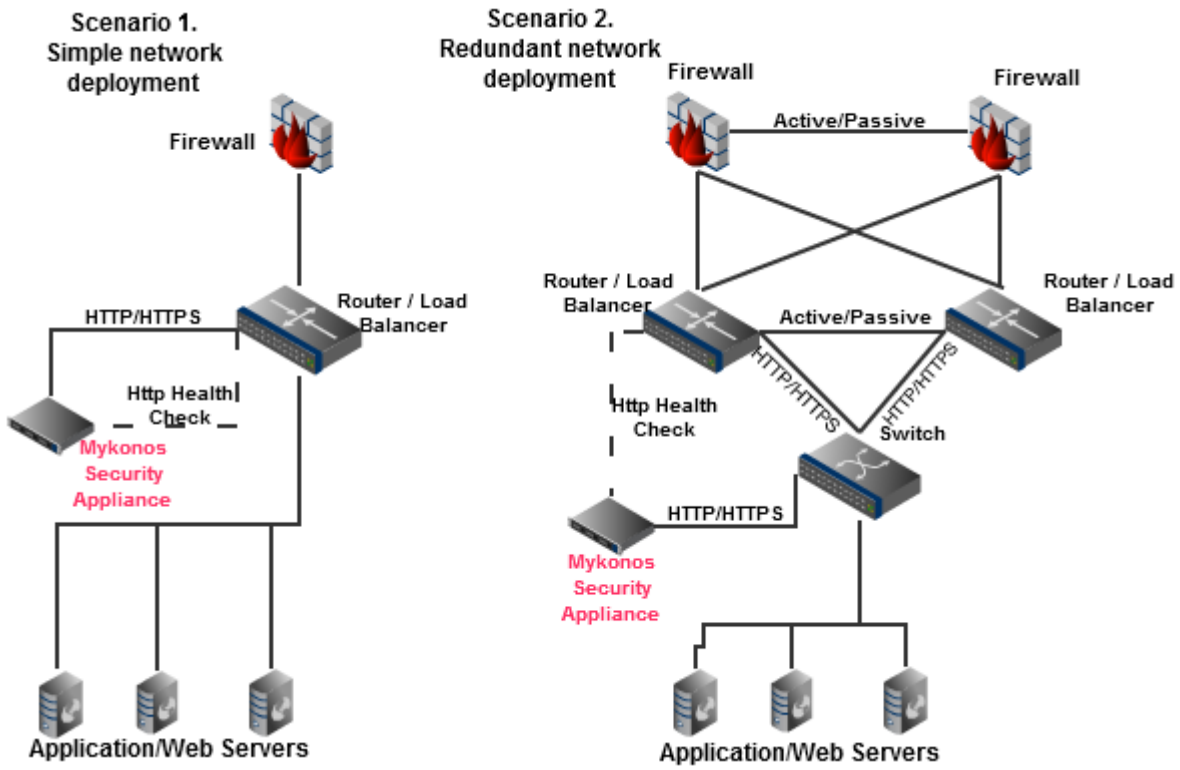
In order to facilitate the unique functionality that MWS provides, it needs to be positioned directly in front of, or in-line with the last device capable of altering or directing user traffic. An example of such device would be a load balancer.



This particular network placement requirement also includes:

- The device parallel with MWS must be able to separate traffic at Layer 7. MWS only accepts HTTP/HTTPS traffic.
- The device, being a load balancer in the above example, should have the capabilities to monitor the MWS, specifically via HTTP. Inability for this device to monitor for an MWS failure would result in a loss of service to web applications being protected.
- If the above requirements are met, the device should mark the MWS as down, thus triggering a failover scenario to divert web application traffic, most likely, directly back to the web application servers.

The two scenarios below describe implementation in a simple network (Scenario 1) as well as in more complex redundant network environment (Scenario 2). However, the final implementation will depend of the actual network and systems topology of the customer.



2. Options for providing a load balanced environment

The MWS can provide the simplest of load balancing for HTTP and HTTPS connections to backend servers via round-robin and least active connections algorithm.

3. SSL traffic consideration

Since it is necessary for Mykonos Web Security to have a complete view into the HTTP traffic details, the device is also capable of terminating SSL certificates or decrypting and encrypting connections en route to the protected web servers. These two modes are aptly named SSL termination and passive decryption.

In the SSL termination mode, MWS serves as an SSL certificate server. All traffic traversing the internet will be secure to and from MWS. Conversely, all traffic traversing between the web servers and MWS will be insecure.

Note

All SSL certificates must be pre-installed on MWS through the security management configuration interface.

In the passive decryption mode, MWS will decrypt all SSL requests and encrypt them once the requests have been processed. It will then forward the requests to the protected web servers. Conversely, the web server responses follow the same procedure, where they are decrypted, processed, and encrypted before being sent back to the requestor.

Note

MWS includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org)

All of the network settings necessary to configure SSL and load balancing for MWS are available through MWS setup menu and is further documented Chapter 3.

4. Limitations

- MWS cannot accept any connections other than HTTP/HTTPS (TCP ports 80 and 443).
- MWS should not be considered a network firewall, thus should not be positioned as an edge device.
- MWS does not support any form of redundancy for network traffic, including link aggregation, or fail open hardware devices.
- MWS should not be placed as a physical, in-line device as there is no fail-open capability.
- MWS doesn't have link aggregation, as such it cannot be connected to mutiple, separate network devices that may assume an IP in an active/passive redundant network configuration.

Chapter 3. Installing Mykonos Web Security

To begin the installation of Mykonos Web Security (MWS) connect the server to your network and power on the system. You will be brought to a login screen. Administrators can login to MWS for the first time with the following credentials:

- mykonos
- mykonosadmin

Important

Administrators should change the default password after logging into MWS for the first time. This can be done through the passwd command from the command line.

1. Configuring MWS for the First Time

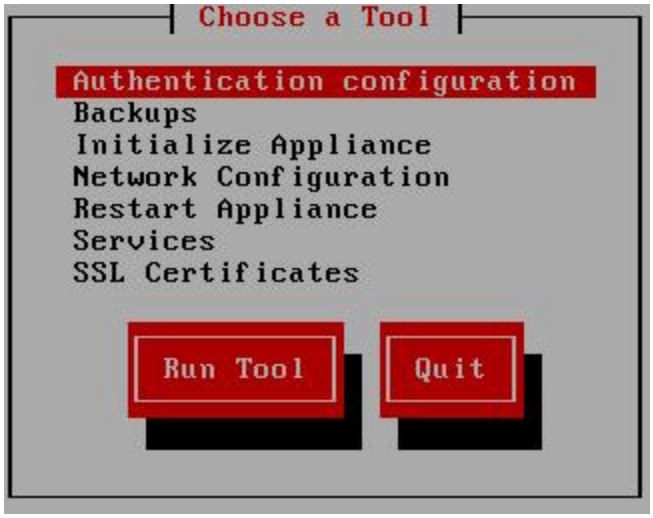
After MWS finishes the intial boot sequence, the configuration menu will launch with the following options:

- Authentication Configuration
- Backups
- Initialize Appliance
- Network Configuration
- Restart Appliance
- Services
- SSL Certificates

2. MWS Setup

The setup menu gives the administrator the ability to set the various configuration items for MWS. If any of these settings need to be changed after the initial setup, the menu system can be invoked at any time by running the following command in a terminal:

```
sudo setup
```



2.1. Network Configuration

The basic requirements for setting up the network interface on the server are to set the hostname and configure the network interfaces. This menu also provides the ability to define network aliases, change DNS details, or restart network services.

Setting the Hostname

To configure the server's network interface, enter the TUI and select the "Network Configuration" menu option. From here enter the "Set Hostname" menu and change the default to the desired hostname.

Configuring Interfaces

Once the hostname is set, open the "Configure Interfaces" menu and enter in the network settings for the interface the server will be using. The following are the configurable options:

- Interface
- Use DHCP
- IP Address
- Netmask
- Gateway
- On Boot

2.2. SSL Certificates

This menu allows administrators to configure SSL for an MWS interface. All certificates must be uploaded to "/home/mykonos/certs" and need to be in PEM (Privacy Enhanced Mail) format.

- Add a new SSL Certificate: Selecting this option will bring up a list of network interfaces, as well as the currently installed certificates that can be associated with them.

- Configure an existing certificate: From this menu administrators can reconfigure the current SSL and private key file associations for any network interface.

Note

SSL on MWS can be enabled through the configuration UI, under "Application Configuration/Traffic/Servers" and "Application Configuration/Traffic/Use SSL Backend" and is covered in more detail in chapter 4.

2.3. Authentication Method

By default MWS uses the local authorization method of the server, but can be configured to use any of the authentication methods used by a Redhat and derivitive distributions. For additional information see:

Important

If the server is configured to use LDAP authentication, then a Mykonos account MUST be created in the LDAP system for users to be able to access the TUI menu system.

- LDAP.
- Kerberos.
- Winbind Authentication.
- Local Authorization.

Note

For more information on configuring authentication visit: http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/ch-Configuring_Authentication.html#sect-The_Authentication_Configuration_Tool-Command_Line_Version

Chapter 4. Configuring Mykonos Web Security

The configuration UI is the menu system responsible for handling all MWS security and application configurations. The UI is divided into three parts; the global configuration menu, the default application menu, and a menu for configuring security for any specific applications that have been defined. The configuration UI can be accessed through a web browser by browsing to "https://MWS_IP:5000".

- Username: mykonos
- Password: mykonosadmin

The configuration user interface (UI) is comprised of the entity tree showing all of the categories of services. After selecting an option from the tree, users will be presented a grid of configuration options for that specific service. In the following example, the global configuration entity has been selected and the general services tab has been opened. From this tab we have set the "Security Engine Enabled" parameter to "True", which enables the security engine service. This is the service responsible for providing the Mykonos Web Security.

1. Basic Configuration

By default, MWS is configured to route all traffic through a default application profile. The only parameter you will need to set to get started will be the target server(s) or virtual IP (VIP) that MWS will proxy for. To get to the parameter, navigate the tree structure on the left hand side down to the "Traffic" section under "Application Configuration". Alternatively, a search for "Servers" using the search box in the top right of the parameters grid. Once the parameter is found, double click on the row to expand the view. Delete the default server entry created during install, or edit it to point to a valid location within your network. Once the necessary changes have been made and the "Servers" entry looks correct in the grid list, click the "Save Configuration" button located in the top left of the Configuration UI. An alert box will appear to let you know that the configuration was successfully saved.

Important

You **MUST** click the "OK" button before saving to ensure that all changes are loaded when the save occurs.

Note

It can take as long as 30 seconds for configuration changes to fully propagate throughout the system.

Note

The Configuration menu, by default will only display the simplest of configuration options. Select the "Avanced" filter to display all of the configuration options.

2. Global Configuration

The global configuration defines the MWS's default actions and has the following configuration menus:

Incident Monitoring (Advanced)	Administrators can set how MWS monitors the security of session tokens.
Session Management (Advanced)	Administrators can configure the session cookies used to track users, as well as the profile naming list.
General Configuration	The security engine and auto responses can be enabled or disabled through this menu option.
Error Handling (Advanced)	Any response codes that will be considered to be unsuccessful requests by MWS.
Default Responses (Advanced)	Allows administrators to upload a custom page that will be returned to the client on a successful request instead of the standard http response page.
Security Configuration	This option is for configuring what data is logged by MWS.
SMTP Configuration	These configuration items setup the SMTP mail service for report and alert emails.
Database Cleanup Service (Advanced)	Administrators can configure how often the database cleanup service runs for sessions, profiles, and incidents through this option.
Security Engine Service (Advanced)	Configures the memory, database, and finger printing used by the security engine.
Session Consolidation Service (Advanced)	Administrators can configure how and when MWS consolidates user sessions.
Auto Response Service	Allows administrators to turn the Auto Response service on or off.
Statistic Service (Advanced)	Configures how MWS logs statisitcal data.
Backup Service	This menu configures the backup service. Administrators can configure the frequency, type, and destination for appliance backups.
Logging (Advanced)	Configures how MWS logs user data. Administrators can configure whether to log request and response data pre and post processing, as well as, change the defualt logging level.

Important

Save each configuration menu item upon change completion as it is possible validation errors may nullify all of the other changes before it can be saved to the system.

2.1. Configuring Backups

To configure the automatic and manual backup options an open session to the configuration UI is needed. The settings are available under the section for "Backups" under "Global Configuration". This will bring up the following screen with these configuration options:

Table 4.1.

Setting	Description
Enable Backups	Whether to perform system backups or not.

Setting	Description
Retention	Number of days that backups will be kept before being deleted.
Frequency	How often backups should be performed.
Push to FTP/SSH	Whether to transfer backups to an FTP/SSH server or not.
FTP/SSH Server	FTP/SSH server that backups will be transferred to.
FTP/SSH Username	Username that will be used for the SSH transfer.
FTP/SSH Password	Password that will be used for the SSH transfer.
FTP/SSH Retries	How many times the transfer will be retried before giving up.

2.2. Configuring Alerts

MWS can be configured to send email alerts to users when an incident of a specified severity is detected. For instance, MWS could be configured to send out an email notification to a specific admin who is on call at the time an incident level of "High" is detected, thus allowing the admin to manually respond (if autorsponses are disabled) quickly to the threat.

Alert contacts are created through the "Alerts" submenu of the "Global Configuration" section of the configuration UI. After selecting the "Alerts" option, users will be brought to the following screen where a new alert contact can be configured by filling in the following fields:

Table 4.2.

Setting	Description
Name	Name of the alert contact.
Email Address	The email address of the alert contact.
Minimum Severity	The minimum severity level that alerts should be created.
Shift Start	The start time of the alert contact's shift.
Shift End	The end time of the alert contact's shift.

To delete a contact simply use the delete button listed next to contact form.

3. Processor Configuration

Each of the MWS processors has it's own set of configurable options that may or may not be applicable to the protected application. Administrators should configure each processor to suit the respective application. Only 2 configurable options exist on each processor. The default setting of "enabled" or changing the setting to "disabled" to turn off the processor. To enable or disable a processor, browse to the desired processor in the left side navigation tree and double click on the processor to expand the menu options. Select the "Procesor Enabled" option and set it to either true (on) or false (off) to turn on/off the respective processor. Once this is done, click the "Save" button located in the top right of the configuration UI. An alert box will appear letting you know that the configuration was successfully saved.

4. Creating an Application Profile

Users should leave the "Default Catch-All Application" to ensure all traffic successfully gets routed to a backend server. New application profiles should be added above it. To create separate profiles for separate applications, an application profile will need to be added for each application server(s) that will be protected by MWS. To add an application profile, right click on the "Applications" tree option in the left hand side navigation menu and select "Add Application". This will add an application profile underneath the "Applications" option once a unique name and appropriate description are added. As with the "Global Application" setting, the following will need to defined:

- A URL pattern that MWS can use route traffic to the correct URL.
- The target servers hosting the URL that MWS will protect.

Important

URL patterns and profiles are processed in the order in which they are created or last arranged. Ensure all new application profiles are added above the default catch-all to avoid conflicts. The order of the "Application Profiles" can be changed by right clicking on a profile and selecting the "Move Up" or "Move Down" options.

Note

If SSL is required for an application, SSL functionality will need to be enabled and the required certificates must uploaded and configured in MWS as described in the "Configuring SSL" section of this guide.

To set the URL pattern of the applicartion server, select the desired application profile under "Applications" from the navigation tree on the left and then double click on the parameter to expand the view. Alternatively, the search option may also be used, though verify that the deisred application is being edited and not the "Global Application". Configure the entries for the URL pattern and server options and click the "Save" button located in the top right of the Configuration UI. An alert box will appear to let you know that the configuration was successfully saved.

4.1. Adding Pages

If there is an application that consists of multiple pages and each page requires unique configuration settings, these can be defined with page specific configurations under the "Applications" menu option. To add a page to an application, double click on the respective application listed in the left hand navigation tree to expand it Right click on the "Pages" option and select "Add Page". A blank page configuration has now been added. This will need to have a defined URI pattern for the page prior to configuring any page specific settings. Once this is done, click the "Save" button located in the top right of the configuration UI. An alert box will appear to validate the configuration was successfully saved.

Traffic	This menu is for defining how network traffic is handled for an application. Default URL patterns for an application page as well as any load balancing or request time out configurations can be configured here as well.
Identification Configuration	This menu defines the MWS settings pertaining to identifying the web server hosting the application. Users can use this menu to assign things like a "Fake Server Name", "Fake Web Root", and a fake "Webmaster Email" for an application.

Incident Monitoring	Used to monitor hackers attempt to manipulate the session token used to track Mykonos proxy sessions
Error Handling	The HTTP error responses to return when fatal or unexpected errors have occured in the reply of servicing a request
Default Responses	Allows the uploading of a custom page or file that will be returned to the client on a successful request instead of the standard http response page.
Security Configurations	This menu specifies the security monitoring of a page. These items can be configured to include the URL in the details of every incident as well as record the request and responses of an incident.
Main Configuration	This is the main menu for enabling and disabling security processors for applications. Any auto responses for applications, such as blocks or warnings, may also be configured here.
Incident Configuration	This menu is used to configure the incident triggers that are used by the MWS.
AJAX Service Configuration	The fake service to expose for the Ajax processor.

4.2. Configuring the URL Pattern

The base URL pattern is used to check and see if a URL belongs to a defined application. The default used here will work for any URL, but for enhanced security regular expression syntax should be changed to reflect the actual defined application's URL.

- URL Pattern: The regular expression patterns used to test request URLs to see if they are for a page. MWS uses perl compatible regular expressions (PCRE) for it's pattern matching. For more information on these please see http://perldoc.perl.org/perlre.html.

4.3. Configuring the Server Health Checks

MWS will check to see if the backend sever(s) are running by submitting an HTTP request to it. By default this is done through an HTTP HEAD request. If the backend server doesn’t accept the HEAD request, the web server will need to be configured to use an accepted method.

Table 4.3.

Setting	Description
Enabled	Turns health checks on or off.
Delay	The amount of time (in milliseconds) between health checks.
Timeout	The amount of time MWS will wait for a health check response.
Fail Count	The number of health checks that can fail before the web server is considered unavailable.
Request	The body of the request used to perform the health check.
Response	The expected response from the server.

4.4. Configuring the Target Servers

In order for Mykonos Web Security to properly route traffic to the web servers, the servers and TCP ports need to be defined. If MWS will be also used as a load balancing solution, the network traffic weight for each server also needs definition.

Table 4.4.

Setting	Description
Server Name	Logical name for the given web server.
Server IP	The IP address of the web server.
HTTP Port	The port on the server that HTTP traffic will be forwarded to.
HTTPS Port	The port on the server that HTTPS traffic will be forwarded to.
Weight	The load balancing weight assigned to the given server.
Backup	Whether the server is a backup server.

4.5. Creating Page Specific Configurations

Since applications can span multiple web pages, MWS allows administrators to define page level security configurations for each individual web page that an application uses. This is done by page specific security configurations that override any global settings previously applied to that page. An example of this might be an application with a separate login screen for an application and it's respective administration console. The application could use a globally defined setting to block users after five failed login attempts, while the administration console might have a specific rule to block users after only three failed login attempts.

To create a new page specific configuration; right click the application in the left navigation tree that the page is a part of. A popup menu will appear; select "Add Page" which will prompt for a logical name to give the new page configuration. Once the name is defined, click "Ok" to create the new page configuration. The page configuration will be now be available under the selected application.

Once the screen has refreshed, navigate through the tree structure to the newly created page configuration. Specify the URL pattern that applies to this configuration. The "URL Pattern" parameter can be found under the "Traffic" category within the "Patterned Based Page" category. Once there, double click the parameter and set the "URI Pattern" attribute to match the URI of the page(s) desired for the new configuration to apply to. After the URI pattern has been updated, click "Ok" and then "Save".

Individual parameters may not be modified under this page configuration. They will only apply to requests with a URI that matches the pattern that has been specified. Any settings applied within the page level configuration will take precedence over global and application level settings.

5. Configuring SSL

To set up Secure Socket Layer (SSL) for an application, all required Certification Authority (CA) signed certificates and keys will need to be moved on to the Mykonos Web Security server and into the "/home/mykonos/certs" and "/home/mykonos/keys" respective directories. Certificates should be in the Privacy Enhanced Mail (PEM) format. Once the certificates have been uploaded to the MWS server they can be loaded into Mykonos Web Security through the setup TUI and configured through the configuration UI.

Within the "Applications/Traffic" sub menu of the configuration UI there is an option on the "Servers" menu option to enable SSL and to define the port for HTTPS traffic to the protected web server.

5.1. Enabling SSL to the Client

To enable SSL between the MWS and the client, upload and configure all of the necessary cerificates from within the TUI. Login to the configuration UI and ensure that the HTTPS field under "Servers" has been populated.

5.2. Enabling SSL to the backend

To enable SSL for the traffic between the MWS server and the protected server, login to the Configuration UI and under "Application Configuration/Traffic" and enable the option for SSL to the web server.

6. Configuring Mail Server

The MWS server utilizes it's own mail server to send alerts and reports to users. It can also be configured to use an external mail server from the "Global/SMTP" section of the configuration UI. The following is a list of the settings that need to be configured to use a mail server other then the MWS server default:

- SMTP Server Address.
- SMTP Server Password.
- SMTP Server Port.
- SMTP Server Username.

Note

When using MWS's default mail server, users should set a valid hostname and ensure that the mail configuration observes all best practices for setting up a mail server.

Chapter 5. Managing the Appliance

1. Restart/Shutdown

To safely reboot or shutdown the Mykonos Web Security server, administrators should use the "Reboot appliance" menu option in the setup TUI accessed through the command line. To access the setup menu use the following command:

```
sudo setup
```

2. System Updates

To update Mykonos Web Security and the server, login to the configuration UI and click the "Updates" button from the button bar across the top of the page. This will bring up a list of updates, if any, that are ready to be installed. The MWS is set to download updates as they are available and can be selected to run individually or all at once.

Note

A backup should always be taken before any update to ensure that if data or settings are lost, there is a current backup to restore to.

If an update does cause a problem that requires that the system be restored from a backup, this can be accomplished through the "Restore from Backup" option in the backups section of the TUI menu.

3. Licensing

To add or update a Mykonos Web Security license, log into the Configuration UI and click the "Licensing" button from the top button bar. This will open the license menu where a valid license can be added by clicking the "Add License" button and entering in a new license key. After entering a license, the user will be asked to accept the Mykonos terms of service. Accepting these terms will complete the licensing process for an Mykonos Web Security.

4. Troubleshooting and Maintenance

4.1. Managing the Services

If for any reason it becomes necessary to check on the status of an MWS service or to start, stop, or restart them. This functionality is provided in the TUI menu under the "Services" menu option.

4.2. Viewing Logs

All of the appliance logs can be found in the "/var/logs/msa". For troubleshooting purposes users may consult the following logs:

- monitor.log

- config.log
- initialize.log
- initialize.log
- security-engine.log
- services.log

The logging levels and content can be configured in the "Global/Logging" section of the configuration UI.

5. Backup and Recovery

By default the appliance is set to create a backup every day, though this setting can be changed through the configuration UI. Backups are stored in "/home/mykonos/backups" and can be used to restore the system by logging into the start up menu and selecting the backup menu option.

5.1. Restoring System Configuration from Backup

If for any reason the appliance suffers a critical error requiring the system to be restored from a backup, this can be accomplished from the TUI setup menu. The setup menu can be launched through the following command from the command line interface: `setup`

In the setup menu there is an option "Restore From Backup" which can be used to restore the system from a retained backup. This menu will be populated with a list of retained system backups that can be used to restore the system to an earlier state along with the following options:

- Restore Latest Backup - This will restore the database using the last saved backup file.
- Reset to Clean State - This option will clean all of the data from the database and set it back to a pristine, or freshly initilized state.

5.2. Backing up and Restoring Console Data

To restore the data that is displayed in the Monitoring Console from a back up, users must use the command line utility. This can be run with the following command: `sudo /usr/sbin/mykonos-db`

Note

If the data is being restored to the Console, a database backup will need to be specified from /opt/msa-current/database or use the "latest" option to restore from the last valid backup.

- backup
- restore (filename)
- restore latest
- restore clean

Chapter 6. Processor Reference

Mykonos Web Security uses a modular approach to securing web applicationa. Each module is responsible for monitoring, detecting and securing a particular aspect of the application and/or individual HTTP request/response. These logical entities are referred to as Security Processors.

Processors are the configurable operators that implement an additional layer of security between the application web servers and the end user. They are also responsible for analyzing the request and response data sent to and from the web server. They also monitor anything from the state of injected tar traps to contents of the headers and body of the HTTP/HTTPS requests and responses. Processors can be managed through the Mykonos Web Security configuration user interface. While some of the operations may be as simple as incrementing a counter, others are far more sophisticated and may alter the request and response data so it is important that administrators configure processors correctly to ensure the web application's security and functionality.

Each processor is monitoring the HTTP stream for specific alterations to what is considered "normal" traffic. These alterations are called "triggers". Each security processor may have several triggers that are responsible for detection. If matched, the processor responsible for handling it will generate a security incident. Incidents vary by complexity, which are further explained in the section below.

1. Complexity Definitions

Complexity definitions comprise ratings of the skills, efforts, and experience necessary to trigger specific incidents. The following is a description of the rating system:

Low (Suspicious)	These incidents can be triggered without any special tools or hacking techniques. It is possible, although unlikely, that a non-malicious user may accidentally exhibit the behavior necessary to trigger this incident. For example, altering the contents of the query string could be the result of a sloppy copy and paste of a URL.
Medium (Passively Malicious)	These incidents can be triggered only with the use of tools or less commonly used browser features. There is no chance that a legitimate user will accidentally trigger these incidents. Incidents of this complexity are primarily triggered by pre-fabricated scripts and scanners, therefore do not reflect the experience of the hacker. It would be easy for a user with malicious intent to trigger an incident of this complexity without actually understanding anything they are doing. An example is that of a hacker making a request with the "OPTIONS http" method. This is usually a step performed by most vulnerability scanners.
High (Directly Malicious)	These incidents can be triggered only with the use of tools or advanced browser features. There is no chance that a legitimate user will accidentally trigger these incidents. Incidents of this complexity generally require a deep understanding of web vulnerabilities in combination with independent specialized tools. It would be fairly difficult for a malicious user to trigger incidents of this complexity without truly understanding the nature of a potential vulnerability. An example being that of a hacker calling a JavaScript AJAX method manually from firebug.
Extreme (Targeted)	These incidents can be triggered only with the use of tools and advanced browser features. There is no chance that a

legitimate user will accidentally trigger these incidents. Incidents of this complexity require a very deep understanding of web vulnerabilities, security tools, and applied hacking techniques. These incidents are extremely difficult for an inexperienced user to trigger. An example might be a hacker using a tool such as John-The-Ripper to crack the encryption used on an MWS presented fake ".htpasswd" file.

2. Security Processors

The Security Processors are separated into four groups:

- Tar Trap Processors
- Activity Processors
- Tracking Processors
- Response Processors

Tar trap processors contain the logic of presenting the fake vulnerabilities and points of interest to the hackers with the goal of exposing the attacker prior to them finding an actual vulnerability in the web application.

Activity processors are the processors that monitor and report any other malicious behavior. These operators watch for malicious activity based on non-injected points of interest. This typically involve monitoring headers, errors, input fields, URL sequences, etc. The ultimate goal is identification of malicious behavior within a valid application stream.

Tracking processors allow for a more advanced tracking of attackers. These processors attempt to collect additional data based on behavioral characteristics and information unique to an attacker's environment. This information makes up "finger prints" that then become the basis for the "hacker database" used in detecting and tracking attackers starting with the very first request they make.

Finally, response processors are the processors that are used for generating responses to the end users. If turned on, these can be used to either manually or automatically (depending on the configuration) respond to a hacker as soon as their activity is detected. In the case of an automated response, these can be tuned to match practically any condition including but not limited to frequency of occurrence, complexity, and types of incidents triggered.

3. Tar Trap Processors

3.1. Access Policy Processor

This processor injects fake permission data into the clientaccesspolicy.xml file of the web application's domain. The fake access policy references a fake service and grants a random domain access to call it. If the service is ever called, or any files are ever requested in the directory the service is supposedly contained in, an incident can be created. Under normal conditions, no user will ever see the clientaccesspolicy.xml file, and therefore be unaware of the URL to the fake service or the directory it resides in. In the cases where a Silverlight object is legitimately requesting clientaccesspolicy.xml from the protected domain in order to access a known service, it will not create an incident, because the service being called is defined with real access directives.

Configuration

Table 6.1.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether or not to enable this process for https traffic.
Advanced			
Incident: Malicious Service Call	Boolean	True	The user manually entered the URL into the browser and accessed the service that way. They did not call the function.
Incident: Service Directory Indexing	Boolean	True	The user asked for a file index on the directory that contains the fake service.
Incident: Service Directory Spider	Boolean	True	The user is issuing requests for resources inside the directory that contains the fake service. Since the directory does not exist, all of these types of requests are unintended and malicious.
Fake Service	String	Random	The fake service the user requested.

Incidents

Incident Name: Malicious Service Call
Complexity: Medium
Default Response: 1x = 5 day block
Cause: MWS adds a fake cookie to the websites it protects. The cookie is intended to look as though it is part of the applications overall functionality, and is often selected to appear vulnerable (such as naming the cookie 'debug' or 'admin' and giving it a numerical or Boolean value). The "Cookie Parameter Manipulation" incident is triggered whenever the fake cookie value changes its value.
Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow", "XSS", "Denial of Service", "Fingerprinting", "Format String", "HTTP Response Splitting", "Integer Overflow", and "SQL injection" among many others. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of

SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because MWS cookie looks just like a normal application cookie, a spider that tests all inputs will eventually test the fake cookie as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake cookie, have also been tested against the rest of the cookies on the site.

Incident Name: Service Directory Indexing
Complexity: Medium
Default Response: 1x = 5 day block
Cause: Originally, embedded HTML technologies such as Flash and Java, were not able to communicate with 3rd party domains. This was a security constraint to prevent a malicious Java or Flash object from performing unwanted actions against a site other then the one hosting the object (for example, a Java applet that brute forces a Gmail login in the background). This limitation was eventually decreased in order to facilitate more complex mash-ups of information from a variety of sources. However to prevent any untrusted websites from abusing this new capability, a resource called the "clientaccesspolicy.xml" was introduced. Now, when a plugin object wants to communicate with a different domain, it will first request "clientaccesspolicy.xml" from that domain. If the file specifies that the requesting domain is allowed to access the specified resource, then the plugin object will be given permission to communicate directly with the 3rd party. The clientaccesspolicy.xml therefore provides a convenient reference for hackers when trying to scope the attack surface of the website. For example, there may be a vulnerable service listed in clientaccesspolicy.xml, but that service may not be referenced anywhere else on the site. So unless the hacker looks at clientaccesspolicy.xml, they would never even know the service existed. MWS will inject a fake service definition into the clientaccesspolicy.xml file in order to identify which users are manually probing the file for information. The "Service Directory Indexing" incident will be triggered if the user attempts to get a file listing from the directory the fake service is supposedly located in.
Behavior: Attempting to get a file listing from the directory where the potentially vulnerable service is located is likely in an effort to identify other unreferenced vulnerable services, or possibly even data or source files used by the service. Such a request represents a "Directory Indexing [http://projects.webappsec.org/w/page/13246922/Directory-Indexing]" attack, and is generally performed while attempting to establish a full understanding of a websites attack surface.

Incident Name: Service Directory Spider
Complexity: Medium
Default Response: 1x = 5 day block
Cause: Originally, embedded HTML technologies such as Flash and Java, were not able to communicate with 3rd party domains. This was a security constraint to prevent a malicious Java or Flash object from performing unwanted actions against a site other then the one hosting the object (for example, a Java applet that brute forces a Gmail login in the background). This limitation was eventually decreased in order to facilitate more complex mash-ups of information from a variety of sources. However to prevent any untrusted websites from abusing this new capability, a resource called the "clientaccesspolicy.xml" was introduced. Now, when a plugin object wants to communicate with a different domain, it will first request "clientaccesspolicy.xml" from that domain. If the file specifies that the requesting domain is allowed to access the specified resource, then the plugin object will be given permission to communicate directly with the 3rd party. The clientaccesspolicy.xml therefore provides a convenient reference for hackers when trying to scope the attack surface of the website. For example, there may be a vulnerable service listed in clientaccesspolicy.xml, but that service may not be referenced anywhere else on the site. So unless the hacker looks at clientaccesspolicy.xml, they would never even know the service existed. MWS will inject a fake service definition into the clientaccesspolicy.xml file in order to identify which users are manually

probing the file for information. The "Service Directory Spidering" incident will be triggered if the user attempts to request a random file inside the directory the fake service is supposedly located in.

Behavior: Requesting a random file from the directory where the potentially vulnerable service is supposedly located is likely in an effort to identify other unreferenced resources. This could include configuration files, other services, data files, etc... Usually an attacker will first attempt to get a full directory index (which only takes one request), but if that fails, the only other technique is to guess the filenames (which could take thousands of requests). Because guessing the file names can take so many requests, there are several publicly available tools that can enumerate over a large list of common file and directory names in a matter of minutes. This type of behavior is an attempt to exploit a server for "Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]" vulnerabilities, and is generally done while the attack is trying to scope the web applications attack surface.

3.2. Ajax Processor

A mistake commonly made by web developers is to consolidate every JavaScript file used by their website into a single file. They then reference that one file from every page on the site, regardless of whether it needs all of the code defined in the file. This is an optimization trick that works, but exposes potential vulnerabilities. The goal is to get the browser to cache all of the external JavaScript, so that you don't need to keep downloading additional code as you navigate the site. Consider the case where one of the pages on the site contains an administrative console written with AJAX technology. In the administrative page, there is a JavaScript file that contains code for managing users of the site (creating user, deleting users, getting user details, etc...). Normally only administrators would visit this page, and they would be the only ones who can see this code. Once all JavaScript on the site is consolidated however, these types of sensitive functions tend to get mixed into the rest of the safer functions. Hackers look for these types of functions in order to find both the administrative page that uses them, as well as exploit the function itself. The goal of this trap is to emulate this common mistake and entice hackers into attempting to exploit the "sensitive looking" function.

Configuration

Table 6.2.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Malicious Script Execution	Boolean	True	The user executed the fake JavaScript function
Incident: Malicious Service Call	Boolean	True	The user manually entered the URL into the browser and accessed the service that way. They did not call the function.

Parameter	Type	Default Value	Description
Inject Script Enabled	Boolean	True	Whether to inject the fake Javascript code into HTML responses.
Service	Configurable	AJAX Service	The fake service to expose

Incidents

Incident Name: Malicious Script Execution
Complexity: High
Default Response: 1x = Slow connection 2-6 seconds and Permanent Block in 10 minutes.
Cause: MWS injects a fake JavaScript file into the websites it protects. This fake JavaScript file is designed to look as though it is intended for administrative use only, but has been mistakenly linked in with non administrative pages. The JavaScript file exposes an AJAX function that communicates with a potentially vulnerable fake service. If the user attempts to invoke this function using a tool like Firebug, this incident will be triggered.

Behavior: It is common practice to create a few single JavaScript files that contain the majority of the code your site needs, and then importing that code into all of the pages. This increases the performance of the site, because the user can download and cache all the JavaScript at once, rather then having to redownload all or some of it again on every page change. However in some cases, developers mistakenly include sensitive administrative functions in with common functions needed by unauthenticated users. For example, a developer might include an "addUser" function into a file that also contains a "changeImageOnHover" function. The "addUser" function may only be called from an administrative UI (behind a login), while the hover image effect would be called on a lot of different pages. Hackers often look through all of the various Javascript files being included on the pages of a website in order to find references to other services that might be vulnerable. Once a function has been identified, the hacker will attempt to find a way to exploit the service the function uses. Because the attacker is actually executing the function instead of attempting to directly communicate with the potentially vulnerable service, this is likely a less sophisticated attack. They are more then likely just trying to determine if the service actually exists, and if they can call it without being authenticated, however depending on the values they supplied as arguments to the function, this could be a number of different attack types, including "Abuse of Functionality [http://projects.webappsec.org/Abuse-of-Functionality]", "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Format String [http://projects.webappsec.org/Format-String]", "Integer Overflows [http://projects.webappsec.org/Integer-Overflows]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", and "SQL Injection [http://projects.webappsec.org/SQL-Injection]"

Incident Name: Malicious Script Introspection
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds and Captcha. 2x = Slow Connection 4-14 seconds and Permanent Block in 10 minutes.
Cause: MWS injects a fake JavaScript file into the websites it protects. This fake JavaScript file is designed to look as though it is intended for administrative use only, but has been mistakenly linked in with non administrative pages. The JavaScript file exposes an AJAX function that communicates with a potentially vulnerable fake service. If the user manually inspects the code of the function and attempts to exploit the service it uses directly (without calling the function itself), this incident will be triggered.

Behavior: It is common practice to create a few single JavaScript files that contain the majority of the code your site needs, and then importing that code into all of the pages. This increases the performance of the site, because the user can download and cache all the JavaScript at once, rather then having to redownload all or some of it again on every page change. However in some cases, developers mistakenly include sensitive administrative functions in with common functions needed by unauthenticated users. For example, a developer might include an "addUser"function into a file that also contains a "changeImageOnHover" function. The "addUser" function may only be called from an administrative UI (behind a login), while the hover image effect would be called on a lot of different pages. Hackers often look through all of the various Javascript files being included on the pages of a website in order to find references to other services that might be vulnerable. Once a function has been identified, the hacker will attempt to find a way to exploit the service the function uses. Unlike the malicious script execution incident, here the attacker has actually dissected the fake AJAX function and attempted to directly exploit the service it uses. This is a more sophisticated attack then actually calling the Javascript function, because it requires that the user understand Javascript logic. Depending on what values they are sending to the service, this could be in an effort to perform any number of exploits, including Abuse of Functionality [http://projects.webappsec.org/Abuse-of-Functionality]", "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Format String [http://projects.webappsec.org/Format-String]", "Integer Overflows [http://projects.webappsec.org/Integer-Overflows]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", and "SQL Injection [http://projects.webappsec.org/SQL-Injection]"

3.3. Basic Authentication Processor

The basic authentication processor is responsible for emulating a vulnerable authentication mechanism in the web application. This is done by publicly exposing fake server configuration files (.htaccess and .htpasswd) that appear to be protecting a resource with basic authentication (a part of the HTTP protocol). To the attacker, the site will appear to be exposing a sensitive administrative script on the site, with weak password protection. As the malicious user identifies the availability of such publicly exposed files, they are walked through a series of steps that emulate exposing an additional piece of information. As the final step, if they end up breaking the weakly authenticated password they will then be considered a high threat.

Note

Note: This processor is applicable to sites that are using Apache as front end web servers. The particular files involved (.htaccess and .htpasswd) a specific to Apache web server.)

Configuration

Table 6.3.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Cracked Authentication	Boolean	True	The user has successfully accessed a fake protected

Parameter	Type	Default Value	Description
			resource using a cracked username and password.
Incident: Directory Configuration Requested	Boolean	True	The user has requested the apache directory configuration file .htaccess.
Incident: Directory Passwords Requested	Boolean	True	The user has requested the apache password file .htpasswd
Incident: Invalid Resource Login	Boolean	True	The user has attempted to login to access the fake file protected by basic authentication, but failed.
Incident: Protected Resource Requested	Boolean	True	The user has requested a fake file which is protected by basic authentication
Authorized Users	Collection	Collection	A list of authorized user accounts.
Protected Resource	Configurable	Protected Resource	The fake protected resource.
Randomization Salt	String	Random	A random set of characters used to salt the generation of code. Any value is fine here.

Incidents

Incident Name: Apache Configuration Requested
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds.
Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess [http://httpd.apache.org/docs/current/howto/htaccess.html] file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured.

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration [http://projects.webappsec.org/Server-Misconfiguration]" weakness that might expose a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", "Path Traversal [http://projects.webappsec.org/Path-Traversal]", or "URL Redirector Abuse [http://

projects.webappsec.org/URL-Redirector-Abuse]" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is related to .htaccess. They are requesting a user database file for a password protected resource defined in .htaccess. This file is generally named ".htpasswd". The user either opened the .htaccess file and found the reference to .htpasswd, or they simply tried .htpasswd to see if anything came back (with or without asking for .htaccess). Either way, this behavior is involved in the establishment of a "Credential/Session Prediction" vulnerability. The request for .htpasswd is usually performed while attempting to establish the scope of the websites attack surface, although sometimes is not performed until trying to identify a valid attack vector.

Incident Name: Apache Password File Requested
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds.
Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess [http://httpd.apache.org/docs/current/howto/htaccess.html] file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources.
In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. MWS will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. These directives allude to the existence of a user database file that contains usernames and encrypted passwords.
The "Apache Password File Requested" incident will trigger in the event that the user requests the fake user database file (generally named .htpasswd).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration [http://projects.webappsec.org/Server-Misconfiguration]" weakness that might expose a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", "Path Traversal [http://projects.webappsec.org/Path-Traversal]", or "URL Redirector Abuse [http://projects.webappsec.org/URL-Redirector-Abuse]" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is related to .htaccess. They are requesting a user database file for a password protected resource defined in .htaccess. This file is generally named ".htpasswd". The user either opened the .htaccess file and found the reference to .htpasswd, or they simply tried .htpasswd to see if anything came back (with or without asking for .htaccess). Either way, this behavior is involved in the establishment of a "Credential/Session Prediction" vulnerability. The request for .htpasswd is usually performed while attempting to establish the scope of the websites attack surface, although sometimes is not performed until trying to identify a valid attack vector.

Incident Name: Invalid Credentials
Complexity: High
Default Response: 1x = Slow Connection 2-6 seconds. 15x = Brute Force Invalid Credentials Incident.
Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess [http://httpd.apache.org/docs/current/howto/htaccess.html] file to provide directory level configuration (such as default 404 messages, password

protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. MWS will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource.

Should the user request the password protected resource, MWS will simulate the correct authentication method defined in .htaccess, and simulate the existence of the fake resource. The "Invalid Credentials" incident will trigger in the event that the user requests the fake password protected file and supplies an invalid username and password (as would be the case if they requested the file in a browser and guessed a username and password at the login prompt).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration [http://projects.webappsec.org/Server-Misconfiguration]" weakness that might expose a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", "Path Traversal [http://projects.webappsec.org/Path-Traversal]", or "URL Redirector Abuse [http://projects.webappsec.org/URL-Redirector-Abuse]" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is referenced only from .htaccess. The fake resource is password protected, and the user has attempted to authenticate with bad credentials. This is most likely in an effort to guess a valid username and password combination, such as "admin:admin", or "guest:guest". It may also be part of a larger brute force attempt, where the attacker tries a long list of possible combinations.

This is a poor method for locating valid usernames and passwords, especially since the user database file .htpasswd is actually exposed (albeit fake). So a brute force attack (represented by a large quantity of this incident type) generally means the attacker is less sophisticated. Because the password protected file is not referenced from anywhere outside of .htaccess, this incident should not happen unless an "Apache Configuration Requested" incident has occurred first. If that is not the case, then the hacker has likely established two independent profiles in MWS. This type of behavior is generally performed when attempting to establish a successful attack vector.

Incident Name: Protected Resource Requested
Complexity: High
Default Response: 1x = Slow Connection 2-6 seconds.
Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess [http://httpd.apache.org/docs/current/howto/htaccess.html] file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources.

In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. MWS will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. Should the user request the password protected resource, MWS will simulate the correct authentication method defined in .htaccess, and simulate the existence of the fake resource. The "Protected Resource Requested" incident will trigger in the event that the user requests the fake password protected file and does not supply a username and password (as would be the case if they requested the file in a browser and canceled the login prompt).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration [http://projects.webappsec.org/Server-Misconfiguration]" weakness that might expose a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", "Path Traversal [http://projects.webappsec.org/Path-Traversal]", or "URL Redirector Abuse [http://projects.webappsec.org/URL-Redirector-Abuse]" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is referenced only from .htaccess. The resource is password protected, but the user has not yet tried to supply credentials. This is most likely in an attempt to see if the password protected file actually exists. Because the password protected file is not referenced from anywhere outside of .htaccess, this incident should not happen unless an "Apache Configuration Requested" incident has occurred first. If that is not the case, then the hacker has likely established two independent profiles in MWS. This type of behavior is generally performed when attempting to establish a successful attack vector.

Incident Name: Basic Authentication Brute Force
Complexity: High
Default Response: 1x = Captcha. 2x = Permanent Block.
Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess [http://httpd.apache.org/docs/current/howto/htaccess.html] file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources. In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. MWS will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. Should the user request the password protected resource, MWS will simulate the correct authentication method defined in .htaccess, and simulate the existence of the fake resource. The "Basic Authentication Brute Force" incident will trigger in the event that the user requests the fake password protected file and repeatedly supplies an invalid username and password (as would be the case if the user was guessing various username and password combinations).

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration [http://projects.webappsec.org/Server-Misconfiguration]" weakness that might expose a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", "Path Traversal [http://projects.webappsec.org/Path-Traversal]", or "URL Redirector Abuse [http://projects.webappsec.org/URL-Redirector-Abuse]" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is referenced only from .htaccess. The fake resource is password protected, and the user has attempted to authenticate with a large number of bad credentials. This is most likely in an effort to guess a valid username and password combination, such as "admin:admin", or "guest:guest". This is a poor method for locating valid usernames and passwords, especially since the user database file .htpasswd is actually exposed (albeit fake). So a brute force attack generally means the attacker is less sophisticated. Because the password protected file is not referenced from anywhere outside of .htaccess, this incident should not happen unless an "Apache Configuration Requested" incident has

occurred first. If that is not the case, then the hacker has likely established two independent profiles in MWS. This type of behavior is generally performed when attempting to establish a successful attack vector.

Incident Name: Password Cracked
Complexity: High
Default Response: 1x = Permanent Block.
Cause: Apache is a web server used by many websites on the internet. As a result, hackers will often look for vulnerabilities specific to apache, since there is a good chance any given website is probably running apache. One such vulnerability involves the use of an .htaccess [http://httpd.apache.org/docs/current/howto/htaccess.html] file to provide directory level configuration (such as default 404 messages, password protected resources, directory indexing options, etc...), while not sufficiently protecting the .htaccess file itself. By convention, any resource that provides directory level configuration should not be exposed to the public. This means that if a user requests .htaccess or a related resource, they should get either a 404 or a 403 error. Unfortunately, not all web servers are configured correctly to block requests for these resources.
In such a scenario, a hacker could gain valuable intelligence on the way the server is configured. MWS will automatically block any requests for the .htaccess resource, and return a fake version of the file. The fake version of the file will contain the directives necessary to password protect a fake resource. The directives will also allude to the existence of a password database file. If the attacker requests the password database file, and then uses a tool such John The Ripper [http://www.openwall.com/john/] to crack one of the encrypted passwords, they will be able to authenticate against the fake protected resource successfully.
Should the user request the password protected resource, and supply a valid username and password combination (as defined in the password database), the "Password Cracked" incident will be triggered.

Behavior: Hackers will often attempt to get the .htaccess file from various directories on a website in an effort to find valuable information about how the server is configured. This is usually done to find a "Server Misconfiguration [http://projects.webappsec.org/Server-Misconfiguration]" weakness that might expose a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]", "OS Commanding [http://projects.webappsec.org/OS-Commanding]", "Path Traversal [http://projects.webappsec.org/Path-Traversal]", or "URL Redirector Abuse [http://projects.webappsec.org/URL-Redirector-Abuse]" vulnerability among others. The fact that an .htaccess file is even exposed is a "Server Misconfiguration" vulnerability in itself. In this specific case, the attacker is asking for a different resource that is referenced only from .htaccess. The fake resource is password protected, and the user has supplied valid authentication credentials. The only way to obtain valid credentials is to either brute force the login (which would be the case if there were excessive numbers of "Invalid Credential" incidents), or to access the fake password database file (usually .htpasswd) and crack one of the encrypted passwords using an encryption cracking tool. This represents the final and most complicated step in a successful "Credential/Session Prediction" exploit, and is usually performed long after the attack surface of the site has been fully scoped. Because the password protected file is not referenced from anywhere outside of .htaccess, this incident should not happen unless an "Apache Configuration Requested" incident has occurred first. Additionally, unless there are excessive numbers of "Invalid Credential" incidents, which would be the case in a brute force attack, the user must have also requested ".htpasswd", and therefore should also have an "Apache Password File Requested" incident.
If either of these incidents is missing, then the hacker has likely established two independent profiles in MWS.

3.4. Cookie Processor

Cookies are used by web applications to maintain state for a given user. They consist of key/value pairs that are passed around in headers and also stored on the client side. Each key/value pair has various attributes that include which domains they are valid for, what paths exist within those domains, as well as security

restrictions and expiration information. Because this is the primary way for a web application to maintain a session, hackers will often try to manipulate cookie values manually in effort to escalate access or hijack someone else's session. All of the attacks applicable to modifying form parameters are also applicable to modifying cookie parameters. It may even be possible, although unlikely, to find an SQL injection flaw in a cookie parameter.

Table 6.4.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether or not to enable this process for https traffic.
Advanced			
Incident: Cookie Parameter Manipulation	Boolean	True	The user modified the value of a cookie which should never be modified.
Cookie	String	Cookie	The fake cookie to use.

Incidents

Incident Name: Cookie Parameter Manipulation
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds and Permanent Block in 10 minutes.
Cause: MWS adds a fake cookie to the websites it protects. The cookie is intended to look as though it is part of the applications overall functionality, and is often selected to appear vulnerable (such as naming the cookie 'debug' or 'admin' and giving it a numerical or Boolean value). The "Cookie Parameter Manipulation" incident is triggered whenever the fake cookie value changes its value.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", and "SQL injection [http://projects.webappsec.org/SQL-Injection]" among many others. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable.

As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because MWS cookie looks just like a normal application cookie, a spider that tests all inputs will eventually test the fake cookie as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake cookie, have also been tested against the rest of the cookies on the site.

3.5. File Processor

When developing websites, administrators will often rename files in order to make room for a newer version of the file. They may also archive older files. A common vulnerability is the case where these older files are left in the web accessible directories, and they contain non-static resources. For example, consider the case where a developer renames shopping_cart.php to shopping_cart.php.bak. If an attacker looks for php files and tries to access all of them with a .bak extension, they may stumble across the backup file. Because the server is not configured to parse .bak files as php files, it will serve the unexecuted script source code to the client. This technique can yield database credentials, system credentials, as well as expose more serious vulnerabilities in the code itself. The goal of this processor is to detect when a user is attempting to find unreferenced files.

Configuration

Table 6.5.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Suspicious File Exposed	Boolean	True	A file which has a suspicious filename is publically available.
Incident: Suspicious Filename	Boolean	True	A file with a filename that contains a suspicious token was requested
Block Response	Configurable	HTTP Response	The response to return when a request is blocked due to a matching suspicious token rule with blocking enabled.
Suspicious Token	Collection	Collection	The configured suspicious extensions.

Incidents

Incident Name: Suspicious File Name Requested
Complexity: Medium
Default Response: 10x = Suspicious File Enumeration Incident.
Cause: MWS has a list of file tokens which represent potentially sensitive files. For example, developers will often rename source files with a ".bck" extension during debugging, and sometimes they forget to delete the backup after they are done. Hackers often look for these left over source files. MWS is configured to look for any request to a file with a ".bck" extension (as well as any other configured extensions), and trigger this incident if the file does not exist. An incident will not be triggered if the file does in fact exist, and the extension is not configured to block the response. This is to avoid legitimate files being flagged as suspicious filenames.

Behavior: There are specific files that many websites host, that contain valuable information for a hacker. These files generally include data such as passwords, SQL schema's, source code, etc... When hackers try to breach a site, they will often check to see if they can locate some of these special files in order to make their jobs easier. For example, if a hacker sees that the home page is called "index.php", they may try and request "index.php.bak", because if it exists, it will be returned as raw source code. This is usually an effort to exploit a "Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]" vulnerability. Automated scanners will generally test all of these types of extensions (.bck, .bak, .zip, .tar, .gz, etc...) against every legitimate file that is located through simple spidering. Because this incident is only created if the file being requested does not actually exist, it does not represent a successful exploit.

Incident Name: Suspicious File Exposed
Complexity: None
Default Response: 10x = Suspicious File Enumeration Incident.
Cause: MWS has a list of file tokens which represent potentially sensitive files. For example, developers will often rename source files with a ".bck" extension during debugging, and sometimes they forget to delete the backup after they are done. Hackers often look for these left over source files. MWS is configured to look for any request to a file with a ".bck" extension (as well as any other configured extensions), and trigger this incident if the extension is configured as illegal. This incident will only be triggered if the file actually exists, and it is configured to be blocked by default. For example, the user might request "database.sql". If the .sql extension is configured to block, and the file actually exists on the server, this incident will be generated. If "database.sql" does not exist, then only a "Suspicious Filename" incident will be created.

Behavior: There are specific files that many websites host, that contain valuable information for a hacker. These files generally include data such as passwords, SQL schema's, source code, etc... When hackers try to breach a site, they will often check to see if they can locate some of these special files in order to make their jobs easier. For example, if a hacker sees that the home page is called "index.php", they may try and request "index.php.bak", because if it exists, it will be returned as raw source code. This is usually an effort to exploit a "Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]" vulnerability. Automated scanners will generally test all of these types of extensions (.bck, .bak, .zip, .tar, .gz, etc...) against every legitimate file that is located through simple spidering. This incident is only triggered when the user requested a file that would otherwise have been successfully returned, if it were not blocked by MWS. For example, the user might request "database.sql" and actually get a 200 response from the server indicating that the file exists and is accessible to everyone. However if the Mykonos appliance is configured to mark the ".sql" extension as illegal, then MWS will block the request and trigger this incident. This prevents the sensitive file from potentially being exposed to an actual malicious user. If this incident occurs, the server administrator should immediately remove the sensitive file or change its permissions so it is no longer publicly accessible.

Incident Name: Suspicious File Enumeration
Complexity: None
Default Response: 1x = Five day block.
Cause: MWS has a list of file tokens which represent potentially sensitive files. For example, developers will often rename source files with a ".bck" extension during debugging, and sometimes they forget to delete the backup after they are done. Hackers often look for these left over source files. MWS is configured to look for any request to a file with a ".bck" extension (as well as any other configured

extensions), and trigger a Suspicious Filename incident if the file does not exist. Should the suspicious filename incident be triggered several times, this incident will then be triggered.

Behavior: There are specific files that many websites host, that contain valuable information for a hacker. These files generally include data such as passwords, SQL schema's, source code, etc... When hackers try to breach a site, they will often check to see if they can locate some of these special files in order to make their jobs easier. For example, if a hacker sees that the home page is called "index.php", they may try and request "index.php.bak", because if it exists, it will be returned as raw source code. This is usually an effort to exploit a "Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]" vulnerability. Automated scanners will generally test all of these types of extensions (.bck, .bak, .zip, .tar, .gz, etc...) against every legitimate file that is located through simple spidering. The first few times a user requests a filename containing a suspicious token, they will only "suspicious filename" incidents. However if they request a large volume of filenames with suspicious tokens, then the "suspicious resource enumeration" incident is generated. This incident represents a user who is actively scanning the site with very aggressive tactics to find unlinked and sensitive data.

3.6. Hidden Input Processor

Many webmasters create forms which post to a common form handling service; using hidden fields to indicate how the service should handle the data. A common hacking technique is to look for these hidden parameters and see if there is any way to change the behavior of the service by manipulating its input parameters. This processor is responsible for injecting a fake hidden input into forms in HTML responses and ensuring that when those values are posted back to the server, they have not been modified.

Configuration

Table 6.6.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Hidden Input Parameter Manipulation	Boolean	True	The user submitted the form and the value of the injected parameter is not what was expected.
Incident: Hidden Input Type Manipulation	Boolean	True	The user submitted the form and the value of the injected parameter is not what was expected. It was also modified to post a file.
Incident: Hidden Input Parameter Manipulation	String	Hidden	The hidden input parameter to inject into forms

Parameter	Type	Default Value	Description
Inject Input Enabled	Boolean	True	Whether to inject hidden inputs into HTML forms

Incidents

Incident Name: Parameter Type Manipulation
Complexity: Medium
Default Response: 1x = Permanent Block.
Cause: MWS inspects outgoing traffic for HTML forms with a "POST" method type. Forms that post to a local URL (within the same domain), will be modified to include a fake hidden input with a defined value. The input is intended to look as though it was always part of the form, and is often selected to appear vulnerable (such as naming the input 'debug' or 'loglevel' and giving it a numerical or Boolean value). The input will however, always be assigned a value that can be represented as a string of characters (in other words, not binary data). The "Parameter Type Manipulation" incident is triggered whenever the fake hidden input is modified from its originally assigned value in order to submit a multipart file.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", and "SQL injection [http://projects.webappsec.org/SQL-Injection]" among many others. Unlike a normal "hidden parameter manipulation" incident, this version is triggered when the user changes the encoding of the form and submits the hidden input as a file post. This is likely in an attempt to either achieve a "Buffer Overflow", or to exploit a filter evasion weakness, that might have otherwise blocked the value being submitted. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because MWS injects several fake inputs, a spider that tests all inputs will eventually test the fake input as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake input, have also been tested against the rest of the inputs on the site.

Incident Name: Hidden Parameter Manipulation
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds. 2x = Logout User. 3x = Clear Inputs.
Cause: MWS inspects outgoing traffic for HTML forms with a "POST" method type. Forms that post to a local URL (within the same domain), will be modified to include a fake hidden input with a defined value. The input is intended to look as though it was always part of the form, and is often selected to appear vulnerable (such as naming the input 'debug' or 'loglevel' and giving it a numerical or Boolean value). The "Hidden Parameter Manipulation" incident is triggered whenever the fake hidden input is modified from its originally assigned value.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", and "SQL injection [http://projects.webappsec.org/SQL-Injection]" among many others. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because MWS injects several fake inputs, a spider that tests all inputs will eventually test the fake input as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It should be assumed that the values tested against the fake input, have also been tested against the rest of the inputs on the site.

Incident Name: Hidden Input Type Manipulation
Complexity: Medium
Default Response: 1x = Permanent Block.
Cause: MWS inspects outgoing traffic for HTML forms with a "POST" method type. Forms that post to a local URL (within the same domain), will be modified to include a fake hidden input with a defined value. The input is intended to look as though it was always part of the form, and is often selected to appear vulnerable (such as naming the input 'debug' or 'loglevel' and giving it a numerical or Boolean value). The input will however, always be assigned a value that can be represented as a string of characters (in other words, not binary data). The "Parameter Type Manipulation" incident is triggered whenever the fake hidden input is modified from its originally assigned value in order to submit a multipart file.

Behavior: Modifying the inputs of a page is the foundation of a large variety of attack vectors. Basically, if you want to get the backend server to do something different, you need to supply different input values (either by cookie, query string, url, or form parameters). Depending on what value the user chose for the input, the attack could fall under large number of vectors, including "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", and "SQL injection [http://projects.webappsec.org/SQL-Injection]" among many others. Unlike a normal "hidden parameter manipulation" incident, this version is triggered when the user changes the encoding of the form and submits the hidden input as a file post. This is likely in an attempt to either achieve a "Buffer Overflow", or to exploit a filter evasion weakness, that might have otherwise blocked the value being submitted. A common practice is to first spider the website, then test every single input on the site for a specific set of vulnerabilities. For example, the user might first index the site, then visit each page on the site, then test every exposed input (cookie, query string, and form inputs) with a list of SQL injection tests. These tests are designed to break the resulting page if the input is vulnerable. As such, the entire process (which can involve thousands of requests) can be automated and return a clean report on which inputs should be targeted. Because MWS injects several fake inputs, a spider that tests all inputs will eventually test the fake input as well. This means that if there is a large volume of this incident, it is likely due to such an automated process. It

should be assumed that the values tested against the fake input, have also been tested against the rest of the inputs on the site.

3.7. Hidden Link Processor

When trying to exploit a site, hackers will often scan the contents of the site in search of directories and files that are of interest. Because this activity is done at the source level, the hacker finds every file referenced, whereas when a user views a website, they can only see the links that are visible according to the HTML. This processor injects a fake link into documents that references a file that looks interesting. The link is injected in such a way that prevents it from being rendered when the browser loads the page. This means that no normal user would ever find/click on the link, but that a scanner or hacker who is looking at the source code likely will.

Configuration

Table 6.7.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Link Directory Indexing	Boolean	True	The user requested a directory index on one of the fake parent directories of the linked file.
Incident: Link Directory Spidering	Boolean	True	The user requested a resource inside the fake directory of the linked file.
Incident: Malicious Resource Requested	Boolean	True	The user requested the fake linked resource.
Hidden Links	Configurable	Hidden Links	The set of hidden links that can be injected into the site.
Inject Link Enabled	Boolean	True	Whether to inject the link into HTTP responses.

Incidents

Incident Name: Link Directory Indexing
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds and 1 Day Block.
Cause: MWS injects a hidden link into pages on the protected web application. This link is not exposed visually to users of the website. In order to find the link, a user would need to manually inspect the source

code of the page. If a user finds the hidden link code in the HTML, and attempts to get a directory file listing from the directory the link points to, this incident will be triggered.

Behavior: A common technique for hackers when scoping the attack surface of a website is to spider the site and collect the locations of all of its pages. This is generally done using a simple script that looks for URL's in the returned HTML of the home page, then requests those pages and checks for URL's in their source, and so forth. Legitimate search engine spiders will do this as well. But the difference between a legitimate spider and a malicious user, is how aggressively they will use the newly discovered URL to derive other URLs. This incident triggers when the user goes beyond just checking the linked URL, but instead also attempts to get a file listing from the directory the URL points to. A legitimate spider would not do this, because it is considered fairly invasive. This activity is generally looking for a "Directory Indexing [http://projects.webappsec.org/Directory-Indexing]"weakness on the server, in an effort to locate unlinked and possibly sensitive resources.

Incident Name: Link Directory Spidering
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds and 5 Day Block in 6 minutes.
Cause: MWS injects a hidden link into pages on the protected web application. This link is not exposed visually to users of the website. In order to find the link, a user would need to manually inspect the source code of the page. If a user finds the hidden link code in the HTML, and attempts to request some other arbitrary file in the same fake directory as the link, this incident will be triggered.

Behavior: A common technique for hackers when scoping the attack surface of a website is to spider the site and collect the locations of all of its pages. This is generally done using a simple script that looks for URL's in the returned HTML of the home page, then requests those pages and checks for URL's in their source, and so forth. Legitimate search engine spiders will do this as well. But the difference between a legitimate spider and a malicious user, is how aggressively they will use the newly discovered URL to derive other URLs. This incident triggers when the user goes beyond just checking the linked URL, but instead also attempts to request one or more arbitrary files inside the same directory as the file referenced by the hidden link. A legitimate spider would not do this, because it is considered fairly invasive.
This activity is generally looking for a "Directory Indexing [http://projects.webappsec.org/Directory-Indexing]" weakness on the server, or a "Predictable Resource Location" [http://projects.webappsec.org/Predictable-Resource-Location] vulnerability, in an effort to locate unlinked and possibly sensitive resources.

3.8. Query String Processor

Hackers tend to manipulate the values of query string parameters in order to get the application to behave differently. The goal of this processor is to add fake query string parameters to some of the links and forms in the page, and verify that they do not get modified when accessed by the user.

Configuration

Table 6.8.

Parameter	Type	Default Value	Description
Basic			

Parameter	Type	Default Value	Description
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Query Parameter Manipulation	Boolean	True	The user manually modified the value of a query string parameter.
Fake Parameters	String	Query String	The collection of fake parameters to add to the links which already have parameters
Inject Parameter Enabled	Boolean	True	Whether to inject query string parameters on urls in HTTP responses.

Incidents

Incident Name: Query Parameter Manipulation
Complexity: Low
Default Response: 3x = Slow Connection 2-6 seconds. 5x = 1 Day Block.
Cause: MWS injects a fake query parameter into some of the links of the protected web site. This query parameter has a known value, and should never change, because it is not part of the actual web application. If a user modifies the query parameter value, this incident will be triggered.

Behavior: Query parameters represent the most visible form of user input a web application exposes. They are clearly visible in the address bar, and can be easily changed by even an inexperienced user. However most users do not attempt to change values directly in the query string, unless they are trying to perform some action the website does not normally expose through its interface, or does not make sufficiently easy. Because it is so easy for a normal user to accidentally change a query parameter, this incident alone is not considered strictly malicious. However depending on the value that is submitted, this could be part of a number of different exploit attempts, including "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", and "SQL injection [http://projects.webappsec.org/SQL-Injection]".

3.9. Robots Processor

The Robots.txt proxy processor is responsible for catching malicious spiders that do not behave in accordance with established standards for spidering. Hackers often utilize the extra information sites expose to spiders, and then use that information to access resources normally not linked from the public site. Because this activity is effectively breaking established standards for spidering, this processor will also identify hackers who are using the information maliciously.

Configuration

Table 6.9.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Malicious Spider Activity	Boolean	True	The user requested a resource which was restricted in the spider rules file, indicating this user is not a good spider, but is spidering the site anyway.
Incident: Spider Policy File Requested	Boolean	True	The user requested the spider rules file robots.txt. This may not be a problem, as good spiders will do this too, but users should never do it.
Incident: Spider Spoofing	Boolean	True	The user has requested the spider configuration file (robots.txt), but supplied a user-agent value that is used by browsers. This is likely an attempt to find information about the resources on the site using a manual browser, or by a spider that is attempting to disguise its true purpose.
Fake Disallowed Directories	String	Random	The path to a fake directory to add to the disallow rules in the robots.txt file. This path should be completely fake and not overlap with actual directories
Known Spiders	Uploaded File	File	The list of known spider host names and IP addresses. This should be a file that contains one host name per line or one IP address per line. If a host name beings with a period, it will match any subdomain

Parameter	Type	Default Value	Description
			(example: .msn.com). If an IP address ends in a period, it will match any IP that starts with the provided value (example: 207.46.)

Incidents

Incident Name: Spider Spoofing
Complexity: Low
Default Response: 1x = Captcha, 2x = Permanent Block;
Cause: One of the standard resources that just about every website should expose is called robots.txt. This resource is used by search engines to instruct them on how to spider the website. Two of the more important directives are "allow" and "disallow". These directives are used to identify which directories a spider should index, and which directories it should stay away from. Good practice for any website is to lock down any resource that should not be exposed. However some web masters simply add a "disallow"statement so that those resources do not get indexed and therefore are never found by users. This technique does not work, because attackers will often access robots.txt and intentionally traverse the "disallow" directories in search of vulnerabilities. So in effect, the listing of such directories is basically pointing hackers in the direction of the most sensitive resources on the site. MWS will intercept requests for robots.txt and either generate a completely fake robots.txt file (if one does not exist), or modify the existing version by injecting a fake directory as a disallow directive. The "Spider Configuration Requested" incident (which is considered informational only) is triggered whenever a user requests robots.txt (including legitimate spiders).

Behavior: Requesting robots.txt occurs in two different scenarios. The first is where a legitimate spider, such as Google, attempts to index the website. In this case, the robots.txt file will be requested, and no requests from that client will be issued to the disallow directories. In the second scenario, a malicious user requests robots.txt and then indexes some or all of the disallow directories. The fact that this incident was triggered does not necessarily mean the user is malicious (which is why it is informational), but it does exclude the user from the general population of average users, because the user is attempting to gain information not intended for a normal user. This type of behavior is generally observed while the client is attempting to establish the overall attack surface of the website (or in the case of a legitimate spider, they are attempting to establish the desired index limitations).

Incident Name: Malicious Spider Activity
Complexity: Medium
Default Response: 1x = Captcha and Slow Connection 2-6 seconds. 6x = 1 Day Block.
Cause: One of the standard resources that just about every website should expose is called robots.txt. This resource is used by search engines to instruct them on how to spider the website. Two of the more important directives are "allow" and "disallow". These directives are used to identify which directories a spider should index, and which directories it should stay away from. Good practice for any website is to lock down any resource that should not be exposed. However some web masters simply add a "disallow" statement so that those resources do not get indexed and therefore are never found by users. This technique does not work, because attackers will often access robots.txt and intentionally traverse the "disallow" directories in search of vulnerabilities. So in effect, the listing of such directories is basically pointing hackers in the direction of the most sensitive resources on the site. MWS will intercept requests for robots.txt and either generate a completely fake robots.txt file (if one does not exist), or modify the existing version by

injecting a fake directory as a disallow directive. The "Malicious Spider Activity" incident is triggered whenever a user attempts to request a resource in the fake disallow directory, or attempts to perform a directory index on the disallow directory.

Behavior: Requesting robots.txt occurs in two different scenarios. The first is where a legitimate spider, such as Google, attempts to index the website. In this case, the robots.txt file will be requested, and no requests from that client will be issued to the disallow directories. In the second scenario, a malicious user requests robots.txt and then indexes some or all of the disallow directories. In this specific case, the user has requested robots.txt to obtain the list of disallow directories, and then started searching for resources in those directories. This activity is performed to find a "Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]" vulnerability. Because spidering a directory tends to be a noisy process (lots of requests), there are likely to be many of these incidents if there are any. The sum of occurrences of this incident represent the type of activity the user is performing to index a directory. The set of URL's for which this incident is triggered, represent the filenames the malicious user is testing for. For example, if they were searching for PDF files that contain stock information, there would be an incident for each filename with a PDF extension they tried to request. There is a very strong chance that if the filename was requested in the disallow directory, it was probably requested in every other directory on the site as well. This type of behavior is generally observed while the client is attempting to establish the overall attack surface of the website (or in the case of a legitimate spider, they are attempting to establish the desired index limitations).

4. Activity Processors

4.1. Custom Authentication Processor

The custom authentication processor is designed to add strong and secure authentication to any page in the protected application. The authentication processor also logs malicious activity like invalid logins and modifying cookies or query parameters.

Configuration

Table 6.10.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
User Accounts	Collection	Collection	A collection of user accounts to use for this processor.
Advanced			
Incident: Auth Cookie Tampering	Boolean	True	The user has modified the cookie used to manage custom authentication, probably in an attempt to expose

Parameter	Type	Default Value	Description
			sensitive information or bypass access restrictions.
Incident: Auth Input Parameter Tampering	Boolean	True	The user has modified the parameters used to manage custom authentication, probably in an attempt to expose sensitive information or bypass the authentication mechanism.
Incident: Auth Invalid Login	Boolean	True	The user has attempted to login but supplied invalid credentials, this could be perfectly normal, but large numbers of this type of incident would indicate a brute force attack.
Incident: Auth Query Parameter Tampering	Boolean	True	The user has modified the query parameters that were submitted when the user was asked to originally login. This is likely in an attempt to probe the authentication mechanism for exploits.
Auth Cookie Name	String	Random	The name of the authentication cookie.
Login Page Timeout	Integer	10 Minutes	The number of seconds a login page can be used before it times out. This is intended to prevent attacks based on watching network traffic. It should be as short as is tolerable.
MD5 Script Name	String	Random	The name of the Javascript resource that contains the MD5 code.
Session Timeout	Integer	1 Hour	The number of seconds a session can be idle before it times out.

Incidents

Incident Name: Auth Input Parameter Tampering
Complexity: Medium
Default Response: 3x = Warn User, 5x = Captcha. 9x = 1 Day Block.

Cause: MWS provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user attempts to manipulate the hidden form parameters used to handle authentication.

Behavior: Manipulating hidden input fields in a form, for whatever reason is generally considered malicious. In this case, since the form is being used to password protect a resource, it is likely that the attacker is trying to bypass the authentication by finding a vulnerability in the authentication mechanism. Depending on the modified value they submit, they could be attempting to launch a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others.

Incident Name: Auth Query Parameter Tampering
Complexity: Medium
Default Response: 1x = Warn User. 2x = 1 Day Block.
Cause: MWS provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user attempts to manipulate the query parameters that were submitted with the original unauthenticated request, after authentication has been completed.

Behavior: Manipulating query parameters after authenticating is not very easy to do without a 3rd party tool, and has no legitimate purpose. As such, this type of behavior is most likely related to a user who is trying to smuggle a malicious payload through a network or web firewall. Depending on the value the user submits for the modified query string, they could be attempting a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others. One interesting note is that the user has actually authenticated in order to cause this incident. As such, it is also likely that the account for which the user authenticated has been compromised and should be updated (with a new password). Although it is possible that the true owner of the account has executed the malicious action, and should therefore potentially be banned.

Incident Name: Auth Cookie Tampering
Complexity: Medium
Default Response: 1x = Warn User, 2x = Captcha. 3x = 1 Day Block.
Cause: MWS provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user attempts to manipulate the cookie used to maintain the authenticated session once the user logs in.

Behavior: Manipulating cookies is not easy to do without a 3rd party tool, and has no legitimate purpose. As such, this type of behavior is most likely related to a user who

is trying to perform a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]" attack, or execute an input based attack such as a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others. One interesting note is that the user has actually authenticated in order to cause this incident. As such, it is also likely that the account for which the user authenticated has been compromised and should be updated (with a new password). Although it is possible that the true owner of the account has executed the malicious action, and should therefore potentially be banned.

Incident Name: Authentication Brute Force
Complexity: Medium
Default Response: 1x = Captcha. 2x = 1 Day Block.
Cause: MWS provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user submits a large volume of invalid username and password combinations.

Behavior: Submitting a single invalid username or password is likely a user typo, and is not necessarily malicious. However it does represent a security event, and a large number of these events may represent a more serious threat such as "Brute Force [http://projects.webappsec.org/Brute-Force]". It is possible however, that the invalid username or password might also be an attack vector targeted at the authentication mechanism such as a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others. This incident is a higher level incident that gets tripped when dozens of "Auth Invalid Login" incidents are created. As such, it does not contain much information about the actual accounts being targeted. If more detail is desired, the underlying "Auth Invalid Login" incidents should be reviewed. These incidents are only suspicious (not considered malicious on their own), so the filtering option will need to be set to show non malicious incidents.

Incident Name: Auth Invalid Login
Complexity: Medium
Default Response: 20x = Auth Brute Force Incident.
Cause: MWS provides the capability of password protecting any URL on the protected site. This means that if a user attempts to access that URL, they will be prompted to enter a username and password before the original request is allowed to be completed. This incident is triggered when a user submits an invalid username or password. This incident alone is not necessarily malicious, as it is possible for a legitimate user to accidentally type their username or password incorrectly.

Behavior: Submitting a single invalid username or password is likely a user typo, and is not necessarily malicious. However it does represent a security event, and a large number of these events may represent a more serious threat such as "Brute Force [http://projects.webappsec.org/Brute-Force]". It is possible however, that the invalid username or password might also be

an attack vector targeted at the authentication mechanism such as a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others. So if the value specified for the username and password does not look like a legitimate username and password (they are too long, or contain unusual characters), then this incident may be more serious. However, even in this case, the user is more likely to submit dozens of invalid credentials (not just one), and there is a different incident for that scenario.

4.2. Cookie Protection Processor

This processor is responsible for protecting a set of application cookies from modification or assignment by the user.

Configuration

Table 6.11.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Protected Cookies	Collection	Collection	The name of the protected cookie.
Protection Start Date	Date	2 years	The date to start enforcing cookie protection. This should be far enough in the future to ensure that all old cookies issued before this processor was activated have expired. Any old cookies issued before this date, which have not expired before this date, will automatically expire.
Advanced			
Protected Cookie Signature Suffix	String	Random	The suffix to add to the protected cookie names when generating a signature cookie. For example, if the protected cookie is PHPSESSID and the suffix is _MX,

Parameter	Type	Default Value	Description
			then the signature for PHPSESSID would be in a cookie named PHPSESSID_MX.
Incident: Client Side Cookie Manipulation	Boolean	True	The user either attempted to modify one of the protected cookies, or attempted to assign a new value.

Incidents

Incident Name: Application Cookie Manipulation
Complexity: Medium
Default Response: 1x = Warn User and Logout User. 2x = 5 Day Block.
Cause: MWS is designed to provide additional protection to cookies used by the web application for tracking user sessions. This is done by issuing a signature cookie any time the web application issues a "protected cookie"(which cookies to protect is defined in configuration). The signature cookie ties the application cookie (such as PHPSESSID) to the Mykonos session cookie. If any of the 3 cookies are modified (Mykonos session cookie, signature cookie, or the actual application cookie), then this incident will be triggered, and the application cookie will be terminated (effectively terminating the users session). This prevents any users from manually creating a session cookie, hijacking another users cookie, or manipulating an existing cookie.

Behavior: Manipulation of cookies is generally performed in order to hijack another user's session. However because cookies represent another type of application input, modifications could also be performed to attempt other exploits. If the modified value resembles a legitimate value for the application cookie, then this is likely a session hijacking attempt. If the cookie contains other values that are clearly not valid, then it is more then likely an attack on generic application inputs such as a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", and "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others.

4.3. Error Processor

Errors and error content play a big part in hacking a website. When a hacker obtains an error message, it provides useful information, the very least of which is that the attacker found a way to do something unintended in the web application and the server executed code to handle it. As such, when a user attempts to hack a website, they frequently induce and receive error messages. Often these error messages are very unusual and are not common when a normal user visits the site. For example, the error code 400 (Bad Request) is returned when the raw data in a request does not follow the HTTP standards. While it is possible to get a 400 error by typing invalid characters into the URL, the majority of these errors are caused by 3rd party software (usually not a browser), improperly communicating with the server. A hacker might for example, manually construct a malicious request and forget to include the "Host" header. The goal of this processor is to record unusual and unexpected errors as incidents.

Configuration

Table 6.12.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Illegal Response Status	Boolean	True	The user issued a request that resulted in an error status code that is considered suspicious and possibly malicious.
Incident: Unexpected Response Status	Boolean	True	The user issued a request that resulted in an unknown error status code and could represent a successful exploit.
100 Continue	Boolean	True	Continue.
101 Switching Protocols	Configurable	Error Status	Switching Protocols.
102 Processing	Configurable	Error Status	Processing.
300 Multiple Choices	Configurable	Error Status	Multiple Choices.
301 Moved Permanently	Configurable	Error Status	Moved Permanently.
302 Found	Configurable	Error Status	Found.
303 See Other	Configurable	Error Status	See Other.
304 Not Modified	Configurable	Error Status	Not Modified.
305 Use Proxy	Configurable	Error Status	Use Proxy.
306 Switch Proxy	Configurable	Error Status	Switch Proxy.
307 Temporary Redirect	Configurable	Error Status	Switch Proxy.
400 Bad Request	Configurable	Error Status	Bad Request.
401 Unauthorized	Configurable	Error Status	Unauthorized.
402 Payment Required	Configurable	Error Status	Payment Required.
403 Forbidden	Configurable	Error Status	Forbidden.
404 Not Found	Configurable	Error Status	Not Found.
405 Method Not Allowed	Configurable	Error Status	Not allowed.
406 Not Acceptable	Configurable	Error Status	Not acceptable.
407 Proxy Authentication Required	Configurable	Error Status	Proxy Authentication Required.
408 Request Timeout	Configurable	Error Status	Request Timeout.
409 Conflict	Configurable	Error Status	Conflict.

Parameter	Type	Default Value	Description
410 Gone	Configurable	Error Status	Gone.
411 Length Required	Configurable	Error Status	Length Required.
412 Precondition Failed	Configurable	Error Status	Precondition Failed.
413 Request Entity Too Large	Configurable	Error Status	Request Entity Too Large.
414 Request-URI Too Long	Configurable	Error Status	Request-URI Too Long.
415 Unsupported Media Type	Configurable	Error Status	Unsupported Media Type.
416 Requested Range Not Satisfiable	Configurable	Error Status	Requested Range Not Satisfiable.
417 Expectation Failed	Configurable	Error Status	Expectation Failed.
418 I'm a teapot	Configurable	Error Status	418 I'm a teapot.
422 Unprocessable Entity	Configurable	Error Status	Unprocessable Entity.
423 Locked	Configurable	Error Status	Locked.
424 Failed Dependency	Configurable	Error Status	Failed Dependency.
425 Unordered Collection	Configurable	Error Status	Unordered Collection.
426 Upgrade Required	Configurable	Error Status	Upgrade Required.
449 Retry With	Configurable	Error Status	Retry With.
450 Blocked by Windows Parental Controls	Configurable	Error Status	Blocked by Windows Parental Controls.
500 Internal Server Error	Configurable	Error Status	Internal Server Error.
501 Not Implemented	Configurable	Error Status	Not Implemented.
502 Bad Gateway	Configurable	Error Status	Bad Gateway.
503 Service Unavailable	Configurable	Error Status	Service Unavailable.
504 Gateway Timeout	Configurable	Error Status	Gateway Timeout.
505 HTTP Version Not Supported	Configurable	Error Status	HTTP Version Not Supported.
506 Variant Also Negotiates	Configurable	Error Status	Variant Also Negotiates.
507 Insufficient Storage	Configurable	Error Status	Insufficient Storage.
509 Bandwidth Limit Exceeded	Configurable	Error Status	Bandwidth Limit Exceeded.
510 Not Extended	Configurable	Error Status	Not Extended.

Incidents

Incident Name: Illegal Response Status
Complexity: Medium

Default Response: 10x 404 = Site Enumeration Incident.
Cause: MWS monitors the various status codes returned by the protected website and compares them to a configurable list of know and acceptable status codes. Some status codes are expected during normal usage of the site (such as 200 - OK, or 403 - Not Modified), but some status codes are much less common for a normal user (such as 500 - Server Error, or 404 - File Not Found). When a user issues a request that results in a status code that is configured as illegal, then this incident will be triggered.

Behavior: In the process of attempting to find vulnerabilities on a web server, hackers will often encounter errors. Just a single error or two is likely not a problem, because even legitimate users accidentally type a URL incorrectly on occasion. However when excessive numbers of unexpected status codes are returned, the behavior of the user can be narrowed down and classified as malicious. The actual vulnerability an attacker is looking for, can be identified through the status codes they are being returned. For example, if the user is getting a lot of 404 errors, they are likely searching for unlinked files ("Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]"). If the user is getting a lot of 500 errors, they may be trying to establish a successful "SQL Injection [http://projects.webappsec.org/SQL-Injection]" or "XSS [http://projects.webappsec.org/Cross-Site+Scripting]" vulnerability.

Incident Name: Unexpected Response Status
Complexity: None
Default Response: 10x 404 = Site Enumeration Incident.
Cause: MWS monitors the various status codes returned by the protected website and compares them to a configurable list of know and acceptable status codes. Some status codes are expected during normal usage of the site (such as 200 - OK, or 403 - Not Modified), but some status codes are much less common for a normal user (such as 500 - Server Error, or 404 - File Not Found). When a user issues a request which results in a status code that is not known and does not have any associated configuration, this incident will be triggered.

Behavior: In the process of attempting to find vulnerabilities on a web server, hackers will often encounter errors. Just a single error or two is likely not a problem, because even legitimate users accidentally type a URL incorrectly on occasion. However when excessive numbers of unexpected status codes are returned, the behavior of the user can be narrowed down and classified as malicious. The actual vulnerability the attacker is looking for can be identified through the status codes they are being returned. For example, if the user is getting a lot of 404 errors, they are likely searching for unlinked files ("Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]"). If the user is getting a lot of 500 errors, they may be trying to establish a successful "SQL Injection [http://projects.webappsec.org/SQL-Injection]" or "XSS [http://projects.webappsec.org/Cross-Site+Scripting]" vulnerability. In the case of this incident, the user is getting an unexpected status code. This is likely because of a bug in the web application which the user has found and is attempting to exploit. The URL this incident is created for, should be reviewed to determine why it would be responding with a non standard status code. If the status code is intentionally non-standard, but is acceptable behavior, then the custom status code should be added to the list of known and accepted status codes in config.

4.4. Header Processor

A useful technique when attacking a site is to determine what software the site is using. This is known as fingerprinting the server. There are many methods used, but the basic idea is to look for signatures that identify various products. For example, it might be a known signature that Apache always lists the "Date" response header before the "Last-Modified" response header. If very few other servers follow this same pattern, then checking to see which header comes first could be used as a means of identifying if Apache is being used or not. Other key methods include looking for "Server" or "X-Powered-By" headers that

actually specify the software being used. The goal of this processor is to eliminate headers as a means of fingerprinting a server.

Important

While the goal of this processor is mainly to prevent fingerprinting, it may also catch some malicious behavior and erroneous behavior in the protected applications (potentially as a result of an exploit). As such, the following incidents are recognized by the processor.

Configuration

Table 6.13.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Duplicate Response Header	Boolean	False	The user provided multiple instances of the same header, and the header does not usually allow multiples.
Incident: Duplicate Request Header	Boolean	True	The application returned multiple instances of the same header, which it is never expected to do.
Incident: Illegal Request Header	Boolean	True	The user provided a request header which is known to be involved in malicious activity.
Incident: Illegal Response Header	Boolean	False	The application returned a response header which it is never supposed to return.
Incident: Missing Request Header	Boolean	True	The user issued a request which is missing a required header.
Incident: Missing Response Header	Boolean	True	The application returned a response which is missing a required header.
Incident: Request Header Overflow	Boolean	True	The user issued a request which contained a header that was longer then the allowed maximum.
Incident: Unexpected Request Header	Boolean	False	The user issued a request which contains

Parameter	Type	Default Value	Description
			an unexpected and unknown header.
Header Mixing Enabled	Boolean	True	Whether this processor should shuffle the order of response headers to avoid exposing identifiable information.
Maximum Header Length	Integer	8192	The maximum allowed length of a header in bytes. If header stripping is enabled, then any headers that exceed this length will be removed from the request before proxying.
Request Header Stripping Enabled	Boolean	True	Whether this processor should strip unnecessary headers in request packets to avoid sending malicious data to the server
Response Header Stripping Enabled	Boolean	True	Whether this processor should strip unnecessary response headers to avoid giving away identifiable information
Known Request Headers	Collection	Collection	A list of known request headers.
Known Response Headers	Collection	Collection	A list of known Response headers

Incidents

Incident Name: Illegal Request Header
Complexity: None
Default Response: None
Cause: MWS monitors all of the request headers included by clients. It has a list of known request headers that should never be accepted. This list is configurable, and by default, includes any headers known to be exclusively involved in malicious activity. Should a user include one of the illegal headers, this incident will be triggered. Because the list of illegal headers is configurable, it cannot be guaranteed that the request that contained the header is strictly malicious, but it does signify that the client is doing something highly unusual.

Behavior: Some HTTP headers can be used in order to get the server to do something it isn't designed to do. For example, the "max-forwards" header can be used to specify how many hops within the internal network the request should make before it is dropped. An attacker could use this header to identify how many network devices are between themselves and the target web server. Because the list of illegal headers is customizable, the type of behavior the header relates to can vary. However this type of behavior is generally performed when scoping the attack surface of the website.

Incident Name: Illegal Response Header
Complexity: None
Default Response: None
Cause: MWS monitors all of the response headers sent to the client from the web application. It has a list of known response headers that should never be returned. This list is configurable, and by default, includes any headers known to compromise the server's identity or security. Should the server return one of the illegal headers, this incident will be triggered. Because the list of illegal headers is configurable, it cannot be guaranteed that the request that contained the header is strictly malicious, but it does signify that something unusual has taken place. This may even represent a hackers successful attempt to exploit a backend service.

Behavior: There is a strict set of HTTP response headers that browsers understand and can actually use. Any headers returned by the server outside of the standard set could potentially expose information about the server or its software. Some headers can even be used to execute more complex attacks. In order to protect the server in the event of a serious issue (such as a "Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]"attack), some headers can be configured as illegal. Because the set is configurable, it is not straight forward as to what the actual header means or what vulnerability it might be targeted at.

Incident Name: Duplicate Request Header
Complexity: None
Default Response: None
Cause: MWS monitors all of the request headers sent from the client to the web application. According to the HTTP RFC, no client should ever provide more the one copy of a specific header. For example, clients should not send multiple "Host" headers. However there are a few exceptions, such as the "Cookie" header, which can be configured to allow multiples. If the user sends multiple headers that are not configured explicitly to allow duplicates, then this incident will be triggered.

Behavior: Sending duplicate headers of the same type can be caused by several different things. It is either an attempt to profile the web server and see how it reacts, an attempt to smuggle malicious data into the headers (because a firewall might not look at subsequent copies of the same header), or possibly just be a poorly programmed web client. In either case, it represents unusual activity that sets the user aside from everyone else. It signifies that the user is suspicious and is doing something average users do not do.

Incident Name: Duplicate Response Header
Complexity: None
Default Response: None
Cause: MWS monitors all of the response headers sent from the server to the client. According to the HTTP RFC, no server should ever provide more the one copy of a specific header. For example, servers should not send multiple "Content-Length" headers. However there are a few exceptions, such as the "Set-Cookie" header, which can be configured to allow multiples. If the server attempts to return multiple headers of the same type, which are not configured explicitly to allow duplicates, then this incident will be triggered.

Behavior: The RFC does not allow for servers to return multiple headers of the same type, with a few exceptions, such as "Set-Cookie". If the server does return duplicates for a header that normally does not support duplicates, then there is either a bug in the web application, or the user has successfully executed a "Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]" attack. In either case, the service located at the URL this incident is triggered for should probably be reviewed for response splitting vulnerabilities or bugs that would cause duplicate response headers to be returned.

Incident Name: Missing Request Header
Complexity: None
Default Response: None
Cause: MWS monitors all of the request headers sent from the client to the server. It also maintains a list of headers which are required for all HTTP requests (such as Host and User-Agent). If one of the required headers is not included in a request, this incident will be triggered.

Behavior: Every legitimate client will always supply specific headers such as "Host" and "User-Agent". If a client does not provide these headers, then the client is likely not a legitimate user. There are several different cases of not legitimate clients, such as hacking tools, manually crafted HTTP requests using something like Putty, or a network diagnostic tool such as nagios. Because there are a few cases that are not necessarily malicious (such as nagios), the incident itself is not necessarily malicious. It does however exclude the user from being a legitimate web browser doing the intended actions allowed by the web application.

Incident Name: Missing Response Header
Complexity: None
Default Response: None
Cause: MWS monitors all of the response headers sent from the server to the client. It also maintains a list of headers which are required for all HTTP responses (such as Content-Type). If one of the required headers is not included in a response, this incident will be triggered.

Behavior: If the server is acting correctly, it should always return all of the required response headers. If it is missing a response header, this is likely due to a bug in the web application, or a successfully executed "Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]"attack. In either case, the service located at the URL this incident is triggered for, should probably be reviewed for either response splitting vulnerabilities, or bugs that would cause abnormal HTTP responses (such as dropping the connection immediately after sending the status code).

Incident Name: Missing Host Header
Complexity: None
Default Response: 1x Host Header = Missing Host Header Incident. 1x User Agent Header = Missing User Agent Header Incident.
Cause: All legitimate web browser submit a Host header with each HTTP request. The host header contains the value entered into the address bar as the server. This could be either the server IP address or the domain name. In either case, it will always be provided. If a user submits a request that does not contain a Host header, this incident will be triggered.

Behavior: Not providing a host header is generally an activity performed when trying to scope the attack surface of the website. Some web servers are configured to host different websites from the same IP address, based on which domain name is supplied. Hackers will often attempt to send a request without a host header to see if the server will serve back a default website. If the default website is not the main website, this may provide additional pages the attacker can attempt to exploit. This could be considered a "Server Misconfiguration [http://projects.webappsec.org/Server-Misconfiguration]" weakness, but may also be a legitimate design choice for the web server and its applications. It does not necessarily expose a vulnerability as long as the default web application is secure. Because all major browsers submit host headers on every request, the user would need to take advantage of a more complex tool, such as a raw data client, or HTTP debugging proxy to manually construct a request that does not have a host header.

As such, this activity is almost always malicious. In a few cases, some legitimate monitoring tools may omit this header, but those tools should be added to the trusted IP list in configuration.

Incident Name: Compound Request Header Overflow
Complexity: None
Default Response: 3x = Compound Header Overflow Incident.
Cause: MWS monitors all of the request headers sent from the client to the server. It has a configured limit that defines how long any individual header is allowed to be. After 3 or more headers are submitted that exceed the limit, this incident will be triggered.

Behavior: While not as common as form inputs or query parameter inputs, some web applications actually use the values submitted in headers within their code base. If these values are treated incorrectly, such as not being validated before being used in an SQL statement, they potentially expose the same set of vulnerabilities a form input might. As such a hacker who is attempting to execute a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]" attack might do so by attempting to provide an excessively long value in a header. They may also use an excessively long header value to craft a complex "SQL Injection [http://projects.webappsec.org/SQL-Injection]" attack. Because the user submitted multiple headers which exceeded the defined limit, the intentions of the user are more likely to be malicious. It is less likely that a poorly crafted browser plug-in would overflow multiple headers, despite the possibility that it might overflow a single one.

Incident Name: Unexpected Request Header
Complexity: None
Default Response: None
Cause: MWS monitors all of the request headers included by clients. It has a list of known request headers that should be accepted. This list includes all of the headers defined in the HTTP RFC document, which means that if any additional headers are passed, it is part of some non standard HTTP extension. Should a user include a non standard header, this incident will be triggered. It is not necessarily a malicious action on its own, but it does signify that the client is unusual in some way (and potentially malicious) and therefore warrants additional monitoring.

Behavior: When attackers are trying to exploit a server, one of the techniques is to attempt to profile what software the server is running. This can be partially accomplished by observing how the server reacts to various types of headers. For example, if the attacker knows that a specific 3rd party web application has a feature where it behaves differently if you send a header "X-No-Auth", then a hacker might send "X-No-Auth" to the site just to see what happens. While this could represent a higher level attack on a specific application; sending non standard headers is more likely part of the hacker's effort to scope the attack surface of the website. This incident alone cannot be deemed malicious because some users have browser plug-ins installed that automatically include non standard headers with requests to some sites. Additionally, some AJAX sites also pass around custom headers as part of their expected protocol.

4.5. Method Processor

GET and POST are two very well known HTTP request methods. A request method is a keyword that tells the server what type of request the user is making. In the case of a GET, the user is requesting a resource. In the case of a POST, the user is submitting data to a resource. There are however, several other supported request methods which include HEAD, PUT, DELETE, TRACE, and OPTIONS. These methods are intended to divide the types of requests into more granular operation. In almost all web application implementations, the PUT, DELETE, TRACE and OPTIONS methods are all left

unimplemented. Unfortunately, some systems provide default implementations for things such as TRACE and OPTIONS. As a result, some administrators accidentally expose unprotected services. Hackers often try these different request methods to identify servers which support them, and therefore may be vulnerable.

Configuration

Table 6.14.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Illegal Method Requested	Boolean	True	The user issued a request using an HTTP method which is considered illegal.
Incident: Unexpected Method Requested	Boolean	True	The user issued a request using a request method other then GET, POST, and HEAD, which resulted in a server error.
DELETE method	Boolean	True	The DELETE method.
GET method	Boolean	True	The GET method.
HEAD method	Boolean	True	The HEAD method.
OPTIONS method	Boolean	True	The OPTIONS method.
POST method	Boolean	True	The POST method.
PUT method	Boolean	True	The PUT method.
TRACE method	Boolean	False	The TRACE method.

Incidents

Incident Name: Unexpected Method Requested
Complexity: None
Default Response: 1x = Slow Connection 2-6 seconds and 5 Day Block in 10 minutes.
Cause: HTTP supports several different "methods" of submitting data to a web server. These methods generally include "GET", "POST", and "HEAD", and less commonly "PUT", "DELETE", "TRACE", and "OPTIONS". MWS monitors all of the methods used by a user when issuing HTTP requests, and compares them to a configured list of known and allowed HTTP methods. If the user submits a request that uses a method which is not in the list of known methods, this incident will be triggered.

Behavior: HTTP methods allow the web server to handle user provided data in different ways. However some of the supported methods are somewhat insecure and should not be supported unless absolutely necessary. In a few cases, methods which are not standard to HTTP are used by 3rd party web applications. When an attacker is looking for a known vulnerability, they may issue requests using some of these custom defined HTTP methods to see if the server accepts or rejects the request. If the server accepts the request,

then the software is likely installed. This type of activity is generally performed when scoping the attack surface of the web application. It is possible that if a 3rdparty web application is legitimately installed and is using custom HTTP methods, that those methods will need to be added to the list of configured HTTP methods so as not to flag users who are using those applications. In either case, because it is possible for this incident to happen without malicious intent, it is considered only suspicious.

Incident Name: Illegal Response Header
Complexity: None
Default Response: 1x = Slow Connection 2-6 seconds and 5 Day Block in 10 minutes.
Cause: MWS monitors all of the response headers sent to the client from the web application. It has a list of known response headers that should never be returned. This list is configurable, and by default, includes any headers known to compromise the server's identity or security. Should the server return one of the illegal headers, this incident will be triggered. Because the list of illegal headers is configurable, it cannot be guaranteed that the request that contained the header is strictly malicious, but it does signify that something unusual has taken place. This may even represent a hackers successful attempt to exploit a backend service.

Behavior: There is a strict set of HTTP response headers that browsers understand and can actually use. Any headers returned by the server outside of the standard set could potentially expose information about the server or its software. Some headers can even be used to execute more complex attacks. In order to protect the server in the event of a serious issue (such as a "Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]"attack), some headers can be configured as illegal. Because the set is configurable, it is not straight forward as to what the actual header means or what vulnerability it might be targeted at.

Incident Name: Unexpected Method Requested
Complexity: None
Default Response: 1x = Slow Connection 2-6 seconds and 5 Day Block in 10 minutes.
Cause: HTTP supports several different "methods"of submitting data to a web server. These methods generally include "GET", "POST", and "HEAD", and less commonly "PUT", "DELETE", "TRACE", and "OPTIONS". MWS monitors all of the methods used by a user when issuing HTTP requests, and compares them to a configured list of known and allowed HTTP methods. If the user submits a request that uses a method which is not in the list of known methods, this incident will be triggered.

Behavior: HTTP methods allow the web server to handle user provided data in different ways. However some of the supported methods are somewhat insecure and should not be supported unless absolutely necessary. In a few cases, methods which are not standard to HTTP are used by 3rd party web applications. When an attacker is looking for a known vulnerability, they may issue requests using some of these custom defined HTTP methods to see if the server accepts or rejects the request. If the server accepts the request, then the software is likely installed. This type of activity is generally performed when scoping the attack surface of the web application. It is possible that if a 3rd party web application is legitimately installed and is using custom HTTP methods, that those methods will need to be added to the list of configured HTTP methods so as not to flag users who are using those applications. In either case, because it is possible for this incident to happen without malicious intent, it is considered only suspicious.

4.6. User Agent Processor

This processor is responsible for detecting when a user manipulates their user agent, or transfers a session from one browser to another.

Configuration

Table 6.15.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Session Porting	Boolean	True	The user has accessed a session from more then one environment. This can happen several ways. Either the user is manually spoofing the User-Agent header, or they manually ported session tracking data from one browser to another. Either case is highly unlikely from a normal user.
User Agent Cache Timeout	Integer	30 Minutes	The number of seconds to cache user agent information for comparison with future requests. If the user waits longer then this period of time between requests with different user agents, the incident will not be detected.

Incidents

Incident Name: Session Porting
Complexity: Medium
Default Response: 1x = Logout User. 2x = Captcha and Logout User. 3x = Warn and Logout User and Slow Connection 2-6 seconds. 4x = Logout User and Slow Connection 4-15 seconds.
Cause: MWS is responsible for tracking information about what software a user is employing to access the protected web applications (eg. A browser, spider, hacking tool, etc...). This information is obtained primarily through a special header called the "user-agent" header. Almost all available software is designed to provide this header when requesting a resource hosted by an HTTP server. In legitimate software, such as mainstream browsers, the value of the user-agent header is not easily configured by the end user. This means that when using the software, it will almost always pass a valid and complete description of what the browser is. In the event that a user modifies or spoofs their user-agent information, the "Session Porting" incident will be triggered. This could happen in one of two ways, either the user manually configured their client to provide a different user-agent value, or the user copied the session cookie from one client to

another. Session cookies are scoped to a single browser, so if the user legitimately changes browser, they will get a new session cookie to go along with their new user-agent.

Behavior: The primary reason for which a user would alter their user-agent string is generally to avoid tracking and detection. The user-agent string contains additional details beyond what browser the user is working with. It also includes information about the operating system, its logged on user, and version information for several popular browser plug-ins. This level of variability not only provides a lot of details on the hacker's environment, but also provides a convenient way to transparently track a user, even without the use of a cookie. As such, many hackers will modify the user-agent value to strip out or spoof any information that is specific to their machine (such as the plug-ins, login information, and possibly even operating system). Additionally, some websites have compatibility issues with specific browser types. This has been a long standing problem with HTML and other client side technologies. As a result, some web servers will actually use the user-agent value to identify what client is requesting a resource, and then return the appropriate compatible version. For example, a user who accesses the site with Internet Explorer might be given a flash enabled site, while a user who accesses the site with FireFox, might be given an SVG enabled site. SVG is a technology that can do much of the same things as flash, but does not require any additional plugins, however Internet Explorer does not support SVG as well as other browsers, so it is not always appropriate. Hackers may attempt to modify their user-agent in order to trick the web server into returning a different version of the site. This is generally done in the beginning of an attack while scoping out the overall websites attack surface. Otherwise if a vulnerability only existed in the IE version of the page, a FireFox hacker might not even know it's there.

5. Tracking Processors

5.1. Etag Beacon Processor

This processor is not intended to identify hacking activity, but instead is intended to help resolve a potential vulnerability in the proxy. Because session tracking in the proxy is done using cookies, it is possible for an attacker to clear their cookies in order to be recognized by the proxy as a new user. This means that if we identify that someone is a hacker, they can shed that classification simply by clearing their cookies. To help resolve this vulnerability, this processor attempts to store identifying information in the browsers JavaScript persistence mechanism. It then uses this information to attempt to identify new sessions as being created by the same user as a previous session. If successful, a hacker who clears their cookies and obtains a new session will be re-associated with the previous session shortly afterwards.

Configuration

Table 6.16.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Session Tampering	Boolean	True	The user has provided a fake ETag value which is not a valid session.

Parameter	Type	Default Value	Description
Beacon Resource	Configurable	Random	The resource to use for tracking.
Inject Beacon Enabled	Boolean	True	Whether a reference to the beacon resource should be automatically injected into HTML responses.
Revalidation Frequency	Integer	3 Minutes	How often in seconds to revalidate the old stored etag and reassociate that session with the current one. This value should not be left too short, because it will cause the browser to constantly rerequest the fake resource and make the tracking technique more visible.

Incidents

Incident Name: Etag Session Spoofing
Complexity: Extreme
Default Response: 1x = Slow Connection 2-6 seconds. 3x = Slow Connection 4-15 seconds.
Cause: The HTTP protocol supports many different types of client side resource caching in order to increase performance. One of these caching mechanisms uses a special header called "E-Tag" to identify when the client already has a valid copy of a resource. When a user requests a resource for the first time, the server has the option of returning an E-Tag header. This header contains a key that represents the version of the file that was returned (ex. an MD5 hash of the file contents). On subsequent requests for the same resource, the client will provide the last E-Tag it was given for that resource. If the server identifies that both the provided E-Tag, and the actual E-Tag of the file are the same, then it will respond with a 403 status code (Not Modified), and the client will display the last copy it successfully downloaded. This prevents the client from downloading the same version of a resource over and over again. In the event that the E-Tag value does not match, the server will return a new copy of the resource and a new E-Tag value. MWS takes advantage of this caching mechanism to store a tracking token on the client. It does this by injecting a fake embedded resource reference (such as an image or a JavaScript file) into some of the pages on the protected site. When the browser loads these pages, it will automatically request the embedded resources in the background. The fake resource that was injected by MWS, will supply a special E-Tag value that contains a tracking token. As the user continues to navigate around the site, each time they load a page that contains a reference to the fake resource, the browser will automatically transmit the previously received E-Tag to the server. This allows MWS to correlate the requests, even if other tracking mechanisms such as cookies are not successful. The E-Tag value returned by the fake resource, which contains the tracking token, is also digitally signed and encrypted, much like MWS session cookie. This prevents a user from successfully guessing a valid E-Tag token, or attempting to provide an arbitrary value without being detected. If an invalid E-Tag is supplied for the fake resource, a "Session E-Tag Spoofing" incident is triggered.

Behavior: There are very few cases where the E-Tag caching mechanism is part of an attack vector, so this incident would almost exclusively represent a user who is attempting to evade tracking or exploit the

tracking method to their advantage. For example, if a user identifies the E-Tag tracking mechanism, they may provide alternate values in order to generate errors in the tracking logic and potentially disconnect otherwise correlated traffic. They may also attempt to guess other valid values in order to correlate otherwise non-related traffic (such as a hacker attempting to group other legitimate users into their traffic).

While this is a highly unlikely attack vector, it could loosely be classified as a "Credential and Session Prediction [http://projects.webappsec.org/w/page/13246918/Credential-and-Session-Prediction]" attack. It is also possible, though unlikely, that once an attacker identifies the dynamic nature of the E-Tag header for the fake resource, they may also launch a series of other attacks based on input manipulation. This could include testing for SQL injection [http://projects.webappsec.org/SQL-Injection], XSS [http://projects.webappsec.org/Cross-Site+Scripting], Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow], Integer Overflow [http://projects.webappsec.org/Integer-Overflows], and HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting] among others. However these would be attacks directly against MWS, and not against the protected web application.

5.2. Client Beacon Processor

The client beacon processor is intended to digitally tag users for later identification by for embedding a tracking token into the client. There are configurable parameters that administrators can use to configure each type of storage mechanisms that are used track malicious users.

Configuration

Table 6.17.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Flash Storage Enabled	Boolean	True	Whether to use the flash shared data API to track the user.
IE UserData Storage Enabled	Boolean	True	Whether to use internet explorers userData storage API to track the user
Local Storage Enabled	Boolean	True	Whether to use Javascript local storage to track the user.
Private Storage Enabled	Boolean	True	Whether to track users between private browsing mode and normal browsing mode in Firefox. A collection of names to use for the Application session cookie.

Parameter	Type	Default Value	Description
Silverlight Storage Enabled	Boolean	True	Whether to use the silverlight storage api to track the user. The silverlight storage API is unique in that it is exposed across all browsers. If this beacon is enabled and the user has silverlight installed, this beacon can track the user even if they switch browsers.
Window Name Storage Enabled	Boolean	True	Whether to use the window.name property of the browser window to track the user.
Incident: Beacon Parameter Tampering	Boolean	True	The user has issued a request to the session tracking service which appears to be manually crafted. This is likely in an attempt to spoof another users session, or to exploit the applications session management. This would never happen under normal usage.
Incident: Beacon Session Tampering	Boolean	True	The user has altered the data stored on the client in an effort to prevent tracking. They have altered the data in such a way as to remain consistent with the same data format. This would never happen under normal usage.
Resource Extensions	Collection	Collection	A collection of resource extensions to use for the processor.
Script Refresh Delay	Integer	1 Hour	The amount of time in seconds to cache the randomly generated set of beacon scripts. After this amount of time, the beacon scripts will change.
Script Variations	Integer	30	The number of random variations of the beacon

Parameter	Type	Default Value	Description
			script to cache, and then to select from on each request.

Incidents

Incident Name: Beacon Session Tampering
Complexity: Medium
Default Response: 1x = 5 day block in 10 minutes.
Cause: MWS uses a special persistent token that inserts itself in multiple locations throughout the client. When a user returns to the site later on, these tokens are transmitted back to the server. This allows the server to correlate the traffic issued by the same user, even if the requests are weeks apart. This incident is triggered when the user manipulates the token data being transmitted to the server on a subsequent visit. They manipulated the data in such a way as to remain consistent with the correct formatting for the token, but the token itself is not valid and was never issued by the server.

Behavior: Attempts to manipulate and spoof the tracking tokens are generally performed when the attacker is trying to figure out what the token is used for and potentially evade tracking. If they are assuming it's used for session management, this might also be a part of a "Credential/Session Prediction [http://projects.webappsec.org/Credential-and-Session-Prediction]" attack. Because the format of the submitted modified token is still consistent with the format expected, this is not likely a generic input attack. It also does not represent any threat to the system, as the modified token is simply ignored.

Incident Name: Beacon Parameter Tampering
Complexity: Low
Default Response: 1x = 5 day block in 10 minutes
Cause: MWS uses a special persistent token that inserts itself in multiple locations throughout the client. When a user returns to the site later on, these tokens are transmitted back to the server. This allows the server to correlate the traffic issued by the same user, even if the requests are weeks apart. This incident is triggered when the user manipulates the token data being transmitted to the server on a subsequent visit. They manipulated the data in such a way as to break the expected formatting for the token.

Behavior: Attempts to manipulate and spoof the tracking tokens are generally performed when the attacker is trying to figure out what the token is used for and potentially evade tracking. Because the format of the token is completely wrong, this is likely a generic input attack, where the user is attempting to find a vulnerability in the code that handles the token. This could include a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others. The content of the manipulated token should be reviewed to better understand what type of attack the user was attempting, however because the tokens are heavily encrypted and validated, this incident does not represent a threat to the security of MWS tracking mechanism.

6. Response Processors

6.1. Block Processor

The block processor is actually a form of auto response. When this processor is enabled, it will allow the security appliance to block a response with "Blocked!" message sent back to the user.

Note

There are no actual triggers for this processor; it is a form of response.

Configuration

Table 6.18.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Block Response	Configurable	HTTP Response	The response to return to the user when they are blocked.

6.2. Request Captcha Processor

The Captcha processor is designed to protect specific pages in a web application against automation. This is done by using a "Captcha" challenge, where the user is required to transcribe random characters from an obscured image or muffled audio file in order to complete the request. The intent is that a human would be capable of correctly answering the challenge, while an automated script with no human intervention would be unable to do so. This assumes that the image is obscured enough that text recognition software is not effective, and the audio file significantly distorted to defeat speech-to-text software. Requiring such user interaction is somewhat disruptive, so it should be utilized only for pages that are prime automation targets (such as contact forms, registration pages, login pages, etc.).

Configuration

Table 6.19.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Bad Captcha Answer	Boolean	True	The user was asked to solve a captcha and

Parameter	Type	Default Value	Description
			entered the wrong value. This could be a normal user error, or it could be the results of failed abuse.
Incident: Captcha Cookie Manipulation	Boolean	True	The user submitted a request and was asked to solve a captcha. They then modified the state cookie used to track captchas, making it invalid. This is likely in an attempt to find a way to bypass the captcha validation mechanism.
Incident: Captcha Directory Indexing	Boolean	True	The user has requested a directory index in the directory that serves the captcha images and audio files. This is likely in an attempt to get a list of all active captchas or to identify how the captchas are generated.
Incident: Captcha Directory Probing	Boolean	True	The user has requested a random file inside the directory that serves the captcha images and audio files. This is likely in an attempt to find an exploitable service or sensitive file that may help bypass the captcha validation mechanism.
Incident: Captcha Disallowed MultiPart	Boolean	True	The user has submitted a multipart form post to the protected page, which has been configured as a disallowed option. This is likely in an attempt to find an edge case the captcha validation mechanism is not expecting.
Incident: Captcha Image Probing	Boolean	True	The user is probing the directory used to serve captcha images. This is likely in an attempt to find hidden files or a way

Parameter	Type	Default Value	Description
			to invoke errors from the captcha serving logic.
Incident: Captcha Parameter Manipulation	Boolean	True	The user has submitted a request with a valid captcha, but they modified the query string parameters. This could be in an attempt to change the output of executing the request without requiring the user to revalidate with another captcha.
Incident: Captcha Request Replay Attack	Boolean	True	The user has attempted to submit the same request multiple times with the same captcha answer. In order words, they solved the captcha once and issued the resulting request multiple times.
Incident: Captcha Request Size Limit Exceeded	Boolean	True	The user has submitted a request to the protected page which contains more data then is allowed. This is may be an attempt to reduce system performance by issuing expensive requests, or it may be an indicator of a more complex attack.
Incident: Captcha Request Tampering	Boolean	True	The user submitted a request and was asked to solve a captcha. They introspected the page containing the captcha and altered the serialized request data (the data from the original request before the captcha prompt). They then submitted a valid captcha using the modified request data. This is likely in an attempt to abuse the captcha system and identify a bypass technique.

Parameter	Type	Default Value	Description
Incident: Captcha Signature Spoofing	Boolean	True	The user submitted a request and was asked to solve a captcha. They introspected the page containing the captcha and provided a validation key from a previously solved captcha. This is likely in an attempt to submit multiple requests under the validation of the first.
Incident: Captcha Signature Tampering	Boolean	True	The user submitted a request and was asked to solve a captcha. They introspected the page containing the captcha and provided a fake validation key. This is likely in an attempt to bypass the captcha validation mechanism.
Incident: Expired Captcha Request	Boolean	True	The user submitted a request and was given a set window of time to solve a captcha. The user solved the captcha and submitted the request for final processing after the window of time expired. This is likely an indication of a packet replay attack, where the user attempts to invoke the business logic of the protected page multiple times under the same captcha validation.
Incident: Mismatched Captcha Session	Boolean	True	The user submitted a request and was asked to solve a captcha. They solved the captcha, but upon submitting the request for final processing, they did so under a different session ID. This is likely due to multiple machines participating in the execution of the site workflow

Parameter	Type	Default Value	Description
			and may indicate a serious targeted automation attack.
Incident: No Captcha Answer Provided	Boolean	True	The user attempted to validate a captcha but did not supply an answer to validate. There is no interface that allows the user to do this, so they must be manually executing requests against the captcha validation API in an attempt to evade the mechanism.
Incident: Unsupported Audio Captcha Requested	Boolean	True	The user has requested an audio version of the captcha challenge, but audio is not supported and there should not be an interface to ask for the audio version. The user is likely trying to find a way to more easily bypass the captcha system.
Bad Request Block Response	HTTP Response	400 HTTP Response	The response to return if the user issues a request that either is too large, or uses multi-part and multi-part is disabled.
Blocked Replay Response	String	Random Value	The response to return if the user attempts to submit the validated request multiple times using the same captcha answer, and that behavior is not allowed.
Captcha Binary Directory	String	Random Value	The name of the directory where captcha images and audio files will be served from. This should not conflict with any actual directories on the site.
Captcha Characters	String	Random Value	The characters to use when generating a random captcha value. Avoid using characters that can be easily mixed

Parameter	Type	Default Value	Description
			up. This set of characters is case sensitive.
Captcha State Cookie Name	String	Random Value	The name of the cookie to use to track the active captchas that have not yet been solved. The cookie is only served to the captcha binary directory.
Captcha Validation Input Name	String	Random Value	The name of the form input used to transmit the captcha validation key. This should be obscure so that users who have not been required to enter a captcha cannot supply bad values to this input to profile MWS.
Maximum Active Captchas	Integer	7	The maximum number of captchas any given user can be solving at any given time. This limit can be overcome, but the majority of users will not be able to. This is primarily for performance, as the more active captchas that are allowed, the larger the state cookie becomes.
Support Audio Version	Boolean	True	Whether an audio version of the captcha is provided to the user. This may be a requirement for accessibility, as vision impaired users would otherwise be unable to solve the captcha.
Watermark	String	Random Value	The text to watermark the captcha with. This can be used to prevent the captcha from being used in a phishing attack. For example, an abuser would not be able to simply display the captcha on a different site and ask a user to solve it. The watermark would tip the user off that the captcha was

Parameter	Type	Default Value	Description
			not intended for the site they are visiting. Use %DOMAIN to use the domain name as the watermark.
Cancel URL	String	None	The URL to redirect the user to if they cancel the captcha. This should not be to the same domain, because the domain is being blocked using a captcha, and therefore, canceling would only redirect to a new captcha. An empty value will hide the cancel button.
Captcha Expiration	Integer	2 minutes	The maximum number of seconds the user has to solve the captcha before the request is no longer possible.
Expired Captcha Response	HTTP Response	400 HTTP Response	The response to return if the user submits a validated request after the captcha has expired. This may happen if the user refreshes the results of the captcha long after they have solved it.
Maximum Request Size	Integer	500kb	The maximum number of bytes in a request before it is considered not acceptable for captcha validation, and will be blocked.
Protected Pages	Collection	None	A collection of protected pages.

Incidents

Incident Name: Captcha Answer Automation
Complexity: Medium
Default Response: 1x = Slow Connection 2-6 seconds and 1 Day Block;
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by

MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides an abnormal volume of bad solutions to the captcha image. For example, the image may have said "Hello", but the user attempted 30 different values all of which did not match "Hello". Because the images can be somewhat difficult to read at times (in order to ensure a script cannot break them), it is not uncommon for a legitimate user to enter the wrong value a few times before getting it right, especially if they are unfamiliar with this type of technique, but after dozens of failed attempts, it is more likely a malicious user.

Behavior: Simply providing a bad solution to the captcha image is not necessarily malicious.Legitimate users are not always able to solve the captcha on the first try. However if a large volume of invalid solutions are provided, then it is more likely that a script is attempting to crack the captcha image through educated guessing and "Brute Force [http://projects.webappsec.org/Brute-Force]".

Incident Name: No Captcha Answer Provided
Complexity: None
Default Response: 1x = Warn User. 2x = 1 Day Block
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user forces the captcha interface to submit the request without a valid captcha solution. There is no way to do this without manipulating the logic that controls captcha protected requests.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved.This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker attempted to submit the captcha protected page without actually solving the captcha. Instead they provided an empty value for the solution parameter. It is not possible to submit an empty solution using the provided captcha interface, so this is almost guaranteed to be a malicious attempt at generating an error and obtaining additional details about the captcha implementation though an "Information Leakage [http://projects.webappsec.org/Information-Leakage]"weakness.

Incident Name: Multiple Captcha Request Overflow
Complexity: None

Default Response: 1 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit dozens of captcha protected requests that exceed the configured maximum for protected request sizes.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved.This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker submitted dozens of extremely large requests, probably in an effort to find a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]" vulnerability, which would produce useful error data and potentially open the server up to further exploitation. They may also be attempting to overload the server and execute a "Denial of Service [http://projects.webappsec.org/Denial-of-Service]" attack.

Incident Name: Unsupported Audio Captcha Requested
Complexity: None
Default Response: 3x = Slow Connection 2-6 seconds and Warn User. 5x = 1 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request the audio version of a captcha challenge when support for audio captchas has been explicitly disabled.

Behavior: Solving an image based captcha is exceptionally difficult and requires a great deal of time and research. Solving an audio captcha however is far less difficult. There are already multiple open source libraries available for translating speech to text. As such, it is often necessary to disable the support of "audio" captchas for critical workflows (such as administrative login dialogs), unless absolutely necessary for accessibility reasons. This incident occurs when the audio captcha has been disabled, but a user is attempting to manually request the audio version of the captcha challenge anyway. The captcha interface

does not expose a link to the audio version unless it is explicitly enabled in configuration, so this would require that the user knows where to look for the audio version, they understand the filename conventions, and they know how to make the request manually to download the file. In either case, if audio captchas are not enabled (through configuration), then this effort will not be successful.

Incident Name: Bad Captcha Answer
Complexity: None
Default Response: 10x = Captcha Answer Automation Incident.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a bad solution to the captcha image. For example, the image may have said "Hello", but the user typed "hfii0"instead. Because the images can be somewhat difficult to read at times (in order to ensure a script cannot break them), it is not uncommon for a legitimate user to enter the wrong value a few times before getting it right, especially if they are unfamiliar with this type of technique.

Behavior: Simply providing a bad solution to the captcha image is not necessarily malicious. Legitimate users are not always able to solve the captcha on the first try. However if a large volume of invalid solutions are provided, then it is more likely that a script is attempting to crack the captcha image through educated guessing and "Brute Force [http://projects.webappsec.org/Brute-Force]".

Incident Name: Mismatched Captcha Session
Complexity: Medium
Default Response: 1x = Warn User, 2x = 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha that was issued for a different session then their own, as might be the case in a script that uses minimal human interaction to solve the captcha's, but everything else is automated.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try and harvest successfully solves captchas from other users on the site. This can be done either by infecting those machines with a virus, or by implanting script into some of the sites pages (possibly through XSS). If this technique is used, then the captcha that is being solved may not have originated from the same session as the user who is submitting the solution. This is a dead giveaway that the user is attempting to defeat the captcha system to automate a specific task.

Incident Name: Expired Captcha Request
Complexity: Low
Default Response: 1x = Warn User. 2x = 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha after the allotted time for solving the captcha has elapsed.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to run expensive image processing algorithms on the captcha image in order to identify what the represented value might be. Additionally, a user might attempt to send the captcha to a warehouse of human captcha solvers. These warehouses specialize in solving large volumes of captchas at a fairly low price (less then a penny per captcha). In either case, it can take several minutes to get the correct captcha answer, and will likely run out the amount of time the user is allowed for solving the captcha. If using a browser, the input would flat out stop accepting answers, but in a scripted scenario, the script will likely try and submit the value anyway, because it is unaware of the expiration. It is possible that this incident would be triggered by a legitimate user, if they were to refresh the page that was produced after the captcha was solved. This would effectively cause the captcha to be reprocessed after the expiration time had been exceeded. As such, this incident on its own is not considered malicious.

Incident Name: Captcha Request Tampering
Complexity: Medium
Default Response: 1x = Warn User. 2x = 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by

MWS.They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha which is correct, but they have modified the parameter containing the original request (which is heavily encrypted to prevent tampering).

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. The parameter that was modified contained the original request data (before the captcha was issued), it is likely that the attacker is attempting to smuggle a malicious payload through MWS without being detected by any network or web firewalls. Because this parameter uses heavy encryption and validation, this type of activity will not produce any useful information or expose any vulnerabilities. Depending on the value they submitted for the original request data, this may also fall under one of the other attack categories involving manipulating general inputs, such as a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]"attack among many others.

Incident Name: Captcha Signature Tampering
Complexity: Medium
Default Response: 1x = Warn User. 2x = 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha which is correct, but they have modified the integrity checking signature passed along with the captcha solution.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. Depending

on the value they submitted for the original request data, this may also fall under one of the other attack categories involving manipulating general inputs, such as a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others.

Incident Name: Captcha Signature Spoofing
Complexity: Extreme
Default Response: 1x = Warn User. 2x = 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user provides a solution to a captcha which is correct, but they have replaced the integrity checking signature passed along with the captcha solution to one that was used in a previous captcha solution.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. This specific incident generally reflects the behavior of a user who is trying to submit a request that would normally be protected by a captcha, but they are trying to trick the system into thinking the captcha was solved correctly, even though it was not. This is generally looking for a "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weakness in the captcha handling mechanism.

Incident Name: Captcha Cookie Manipulation
Complexity: Medium
Default Response: 1x = Warn User. 2x = 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious

user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user alters the cookies used to maintain captcha state.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker modified a cookie that is used to maintain the state of the captcha. The cookie is heavily encrypted, but the attacker may be attempting to establish a way of either identifying what the value of the captcha is algorithmically (by analyzing the cookie value), or they may be attempting to assign a value to the captcha. In either case, this activity generally indicates a user who is trying to find a way to bypass the captcha. Depending on the value they submitted for the original request data, this may also fall under one of the other attack categories involving manipulating general inputs, such as a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", or "SQL injection [http://projects.webappsec.org/SQL-Injection]" attack among many others.

Incident Name: Captcha Image Probing
Complexity: Low
Default Response: 1x = Warn User. 2x = 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request a captcha image file for a request that is not being protected by a captcha.

Behavior: In order to find a way to bypass the captcha mechanism, attackers will often attempt to collect a large number of captcha images for offline analysis. If the attacker can find a pattern in how the captcha images are issued, or how the filename relates to the value in the image, then they can effectively bypass the captcha mechanism at will. In this case, the attacker is guessing arbitrary captcha image filenames, but is attempting to keep the format of the names consistent with known captcha image URL's. Because the filename used and the values in the image have no correlation, this technique will not be successful and will simply waste the attacker's time and resources.

Incident Name: Captcha Request Size Limit Exceeded
Complexity: None
Default Response: 10x = Multiple Captcha Request Overflow Incident.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit a captcha protected request that contains a request body larger then the configured maximum.

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated or if an unexpected outcome can be achieved. This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker submitted an extremely large request, probably in an effort to find a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]" vulnerability, which would produce useful error data and potentially open the server up to further exploitation. This incident is not necessarily malicious on its own, as it is possible for a normal user to submit a value that is larger then the configured maximum, especially if the configured maximum is small, or if the form protected by the captcha allows file posts.

Incident Name: Captcha Disallowed Multipart
Complexity: Medium
Default Response: 10x = Multiple Captcha Disallow Multipart Incident.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit a captcha protected request that contains a binary file, and the captcha is explicitly configured to not allow binary file submission (it has been configured to disallow multi-part form submissions).

Behavior: When a hacker is attempting to establish an automated script that is capable of defeating the captcha, they may use various different techniques. One of these techniques is to try changing various values used by the web application in the captcha mechanism in an effort to see if an error can be generated, or an unexpected outcome can be achieved.This type of probing and reverse engineering is generally performed by advanced hackers. In this specific case, the attacker submitted a binary file in the request that is being protected. The captcha in this case has been explicitly configured to not allow Multi-Part form submissions, so this represents unexpected and undesired activity. Using Multi-Part forms, the attacker can more easily accomplish a "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]"attack, which would produce useful error data and potentially open the server up to further exploitation. Additionally, some web applications do not handle the encoding used for multi-part forms gracefully, so error information may also be obtained from conflicts arising from the submission type. This is not necessarily a malicious incident on its own, because it is possible that the user is legitimately submitting a multi-part form, and just happened to have the captcha activated during the submission.However this is a very rare case, and still represents a somewhat suspicious client.

Incident Name: Captcha Directory Indexing
Complexity: Low
Default Response: 1x = Slow Connection 2-6 seconds and 1 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request a directory index from the same fake directory as the captcha images are being served from.

Behavior: When attempting to either bypass the captcha mechanism, or find a vulnerability in the server, attackers will often try finding unlinked resources throughout the web site. The captcha mechanism uses a fake directory in order to serve the images and audio files that contain the captcha challenge. If the attacker is requesting an arbitrary file within the same fake directory, they are likely trying to find a "Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]" vulnerability. In this specific case, the attacker is attempting to get a full file listing of everything inside the captcha directory. This could potentially be used to get a massive list of all active captcha URL's, or to find resources that are used in the creation of captcha challenges. The directory index will not be allowed, so this does not actually provide the attacker with any useful information.

Incident Name: Captcha Directory Probing
Complexity: Low
Default Response: 1x = Warn User. 2x = Slow Connection 2-6 seconds and 5 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because

it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to request an arbitrary file (not a captcha image, but something else) from within the same fake directory as the captcha images are being served from.

Behavior: When attempting to either bypass the captcha mechanism, or find a vulnerability in the server, attackers will often try finding unlinked resources throughout the web site. The captcha mechanism uses a fake directory in order to serve the images and audio files that contain the captcha challenge. If the attacker is requesting an arbitrary file within the same fake directory, they are likely trying to find a "Predictable Resource Location [http://projects.webappsec.org/Predictable-Resource-Location]" vulnerability. For example, the attacker might be trying to find a source file in the captcha serving directory in hopes of actually being able to get the source code behind how captcha images are generated. Because the directory is fake, the attacker will never find any of the resources they are looking for.

Incident Name: Captcha Parameter Manipulation
Complexity: Low
Default Response: 5x = Multiple Captcha Parameter Manipulation Incident.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit multiple solutions for multiple captchas, but they keep modifying the query parameters that were submitted with the original requests. For example, if the user submitted a "add product to cart" request, and one of the query parameters was the item to add, this incident would be triggered if after solving the captcha, the value of that query parameter was modified to some other value, and this modification happened dozens of times.

Behavior: Because captcha's prevent automation, attackers will sometimes try and find ways to abuse the technique used to request the captcha in order to exploit the site. For example, if the attacker can find a way to submit the same solution over and over again, but have the web application perform a different action each time, they may be able to solve the captcha once and still automate the resulting workflow. In this case, the attacker changed a query parameter that was submitted with the original request. They submitted the original request, solved the captcha, changed the query parameter, and then resubmitted the solved captcha request. In some cases, this might cause the web application to execute a different operation based on the difference in query parameter values. For example, if the protected workflow is "add product to

cart" on a shopping site, then the attacker might attempt to submit the same solved captcha repeatedly, but change the product ID that is being added on each request. This might allow them to automate the addition of products to a shopping cart, after solving only one captcha challenge. The captcha mechanism does not allow the modification of query parameters after the original request has been submitted, so this type of activity will not be successful.

Incident Name: Captcha Request Replay
Complexity: Medium
Default Response: 1x = Warn User. 2x = 1 Day Block.
Cause: A captcha is a special technique used to differentiate between human users, and automated scripts. This is done through a Turing test, where the user is required to visually identify characters in a jumbled image and transcribe them into an input. If the user is unable to complete the challenge in a reasonable amount of time, they are not allowed to proceed with their original request. Because it is nearly impossible to script the deciphering of the image, automated scripts generally get stuck and cannot proceed. Additionally, an audio version is optionally available to allow users who have a visual handicap to complete the captcha successfully. Captchas are used in two different ways by MWS. They can be explicitly added to any workflow within the protected web application (such as requiring a captcha to login, or checkout a shopping cart), and they can be used to test a suspicious user before allowing them to continue using the site (similar to blocking the user, but with a way for the user to unblock themselves if they can prove they are not an automated script). Captchas are generally used to resolve "Insufficient Anti-Automation [http://projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation]" weaknesses in the protected web application. Regardless of which type of captcha is being used, this incident is generated when the user attempts to submit a captcha solution multiple times and "replay" is explicitly disabled for the captcha being used.

Behavior: Because captcha's prevent automation, attackers will sometimes try and find ways to abuse the technique used to request the captcha in order to exploit the site. For example, if the attacker can find a way to submit the same solution over and over again, they may be able to solve the captcha once and still automate the resulting workflow. This is sometimes considered legitimate behavior (as would be expected if the user refreshed the browser after submitting a successful captcha), however in many cases, such functionality would make the captcha significantly less effective at preventing automation. In this case, the attacker resubmitted a request that had already been successfully validated through a captcha, and "replay" was explicitly disabled for the captcha. This is not necessarily a malicious incident on its own, because the user may have accidentally refreshed the browser, however multiple attempts would definitely represent malicious intent. An example of where a captcha's "replay" could cause a problem is on a gaming site, where the user is adding fake "money" to their account. In order to add the fake money, they must solve the captcha. This workflow is protected with a captcha, because if a user could automate the process, they would be able to add unlimited funds to their account. If an attacker were able to solve the captcha once, and continuously resubmit the resulting request, they could effectively add funds over and over again without resolving a new captcha. This would then allow for automation. Replay attackers are less of a problem if the web application being protected already has a method of preventing the same request from being submitted accidentally multiple times. Such would be the case if the web application maintained state information for the given session, and recorded the operation after it was successful, then used that state information to prevent a future occurrence of the operation.

6.3. CSRF Processor

The CSRF processor is intended to prevent cross site request forgery (CRSF) attacks. This is done by taking advantage of the fact that CSRF attacks generally do not render the page they are targeting. For example, a forum might have an image in a post that references the change password service of facebook.

When a user views the post, if they are logged into facebook, their password will be changed. However since the page returns HTML, and the reference is an image tag, the HTML will not be rendered. So to block the attack, the processor will intercept any request with a 3rd party referrer and return an automatic redirect HTML page. The actual request will not reach the backend server until after the redirect (which will change the referrer to a local referrer). If someone references a service in an image tag, the redirect will not take place, and the request will never reach the backend server.

Configuration

Table 6.20.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: CSRF Parameter Tampering	Boolean	True	The user tampered with the parameters used by the security engine to prevent CSRF on requests that have an untrusted 3rd party referer. This is likely in an attempt to find a vulnerability in the CSRF protection mechanism.
Incident: CSRF Remote Script Inclusion	Boolean	True	The user has accessed an untrusted 3rd party website which contains an embeded script reference to the protected application. While the user may not be malicious, this represents a CSRF attack from the untrusted website against the protected application. Because the attack was not successful, it is likely being executed by the user who is attmepting to construct the attack vector.
Block Response	Configurable	HTTP Response	The response to return if the CSRF mechanism cannot complete the request due to errors or tampering.

Parameter	Type	Default Value	Description
Remote Script Resource	Configurable	Random	The fake resource to request if the page is being loaded as a remote script on a third party domain. This is primarily for detection of the attack and can be any fake resoures as long as it does not actually exist on the server.
Trusted Domains	Collection	None	A collection of domains that will be trusted.

Incidents

Incident Name: CSRF Parameter Tampering
Complexity: Medium
Default Response: 10x = Multiple CSRF Parameter Tampering Incident.
Cause: MWS protects against CSRF attacks by using a special interception technique. When a request comes in to MWS, the referrer is checked. In the event that there is a 3rd party referrer (the user was following a link from another site), the interception mechanism kicks in. This involves returning a special page to the user that validates that the user is intentionally requesting the resource. If the validation is successful, the user is transparently redirected to the original resource they requested. If the validation fails, the user is then instructed to manually confirm their intentions, or return to the page they came from (to prevent the CSRF attack from working). In most cases, a valid CSRF attack would function in such a way as to hide this manual confirmation step, so the user would probably never see it (e.g. if the URL was loaded using an image HTML tag, then the resulting HTML confirmation step would not render, because its HTML, not an image). This incident is triggered when a user submits a request with a 3rd party referrer, and then manipulates the code of the CSRF interception page to alter the original data that was submitted. For example, they submit a request that looks like a CSRF attack (has a 3rd party referrer), and then use a tool like Firebug to edit the query string parameters that would be sent to the server after they manually allowed the request on the CSRF intercept page.

Behavior: CSRF attacks are generally two-phase. The first phase involves the attacker establishing a functional CSRF attack. This could take quite a while and involves the attacker making requests to the protected site, trying all different types of CSRF techniques. The second phase is when the attacker injects the successful CSRF vector into a public website. In the second phase, legitimate users are visiting the public website and unknowingly executing the CSRF attack in the background. It is not useful to flag the victims of the CSRF attack as hackers, because they may not even know what is going on. However it is useful to flag the original attack vector establishment, because it may shed light on who created the "CSRF [http://projects.webappsec.org/Cross-Site-Request-Forgery]"attack. This incident reflects a user who is manipulating the CSRF prevention mechanism, likely in an attempt to find a way to get around it.As such, if a user has this incident, they are probably trying to establish a CSRF attack, and careful attention should be paid to the values they are changing the parameters to and which URL is being requested (this will help identify what the user is trying to attack).

Incident Name: Multiple CSRF Parameter Tampering
Complexity: Extreme
Default Response: 10x = Multiple CSRF Parameter Tampering Incident.

Cause: MWS protects against CSRF attacks by using a special interception technique. When a request comes in to MWS, the referrer is checked. In the event that there is a 3rd party referrer (the user was following a link from another site), the interception mechanism kicks in. This involves returning a special page to the user that validates that the user is intentionally requesting the resource. If the validation is successful, the user is transparently redirected to the original resource they requested. If the validation fails, the user is then instructed to manually confirm their intentions, or return to the page they came from (to prevent the CSRF attack from working). In most cases, a valid CSRF attack would function in such a way as to hide this manual confirmation step, so the user would probably never see it (e.g. if the URL was loaded using an image HTML tag, then the resulting HTML confirmation step would not render, because its HTML, not an image). This incident is triggered when a user submits dozens of requests with a 3rd party referrers, and then manipulates the code of the CSRF interception page to alter the original data that was submitted. For example, they submit a bunch of requests that look like CSRF attacks (they have 3rd party referrers), and then use a tool like Firebug to edit the query string parameters that would be sent to the server after they manually allowed the requests on the CSRF intercept page.

Behavior: CSRF attacks are generally two-phase. The first phase involves the attacker establishing a functional CSRF attack. This could take quite a while and involves the attacker making requests to the protected site, trying all different types of CSRF techniques. The second phase is when the attacker injects the successful CSRF vector into a public website. In the second phase, legitimate users are visiting the public website and unknowingly executing the CSRF attack in the background. It is not useful to flag the victims of the CSRF attack as hackers, because they may not even know what is going on. However it is useful to flag the original attack vector establishment, because it may shed light on who created the "CSRF [http://projects.webappsec.org/Cross-Site-Request-Forgery]"attack. This incident reflects a user who is manipulating the CSRF prevention mechanism, likely in an attempt to find a way to get around it. As such, if a user has this incident, they are probably trying to establish a CSRF attack, and careful attention should be paid to the values they are changing the parameters to and which URL is being requested (this will help identify what the user is trying to attack).

Incident Name: CSRF Remote Script Inclusion
Complexity: Extreme
Default Response: 1x = Captcha. 2x = 1 Day Block.
Cause: MWS protects against CSRF attacks by using a special interception technique. When a request comes in to MWS, the referrer is checked. In the event that there is a 3rd party referrer (the user was following a link from another site), the interception mechanism kicks in. This involves returning a special page to the user that validates that the user is intentionally requesting the resource. If the validation is successful, the user is transparently redirected to the original resource they requested. If the validation fails, the user is then instructed to manually confirm their intentions, or return to the page they came from (to prevent the CSRF attack from working). In most cases, a valid CSRF attack would function in such a way as to hide this manual confirmation step, so the user would probably never see it (e.g. if the URL was loaded using an image HTML tag, then the resulting HTML confirmation step would not render, because its HTML, not an image). This incident is triggered when a user accesses a page on a 3rd party website which contains a Javascript tag that loads content from the protected site. This would normally represent a victim of a CSRF attack, but because CSRF attacks are blocked, an attacker is unlikely to execute such an attack. Therefore, it is more probable that the attacker is testing a possible vector to see if it will work and encountering this incident..

Behavior: CSRF attacks are generally two-phase. The first phase involves the attacker establishing a functional CSRF attack. This could take quite a while and involves the attacker making requests to the protected site, trying all different types of CSRF techniques. The second phase is when the attacker injects the successful CSRF vector into a public website. In the second phase, legitimate users are visiting the public website and unknowingly executing the CSRF attack in the background. It is not useful to flag the

victims of the CSRF attack as hackers, because they may not even know what is going on. However it is useful to flag the original attack vector establishment, because it may shed light on who created the "CSRF [http://projects.webappsec.org/Cross-Site-Request-Forgery]"attack. While this incident would potentially be fired for any victims of a CSRF attack, CSRF attacks are blocked by this processor, so it is unlikely that an attacker would ever actually try to use the vector against legitimate users. As such, it is far more likely that the attacker is still in the first phase and trying to uncover a successful CSRF vector. Because of this, careful attention should be paid to the URL that is being requested (this will help identify what the user is trying to exploit).

6.4. Header Injection Processor

This processor provides the header injection counter response. It allows an extra custom header to be defined that is injected into a suspected hacker's requests to allow custom handling.

Note

There are no actual triggers for this processor; it is a form of response.

Configuration

Table 6.21.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Default Header Name	String	Random	The default header name to use if one is not specified in the response configuration.
Default Header Value	String	True	The default header value to use if one is not specified in the response configuration.

6.5. Force Logout Processor

This processor provides the force logout counter response. It strips out and invalidates the users session tokens logging them out of the site.

Note

There are no actual triggers for this processor; it is a form of response.

Configuration

Table 6.22.

Parameter	Type	Default Value	Description
Basic			
Enable Processor	Boolean	True	Whether traffic should be passed through this processor.
Application Session Cookie	Collection	Collection	A collection of names to use for the Application session cookie
Advanced			
Clear Session Cookies	Boolean	False	Whether to clear any terminated session cookies from the malicious users browser. This may help the user identify why they are getting logged off, so unless the application has code on the client that reads the session cookie value, or the cookie is used in traffic not protected by the mykonos appliance, this option should be turned off.

6.6. Strip Inputs Processor

This processor is used to transparently remove all user input from requests being issued to the server. This response will make the web application, or the client accessing it, to appear broken from the users perspective. The website will also take on a much smaller attack surface should the client be a vulnerability scanner.

Note

There are no actual triggers for this processor; it is a form of response.

Configuration

Table 6.23.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			

Parameter	Type	Default Value	Description
None	None	None	None

6.7. Slow Connection Processor

The slow connection processor is designed to introduce large delays in requests issued by malicious traffic without impacting the performance of legitimate users.

Note

There are no actual triggers for this processor; it is a form of response.

Configuration

Table 6.24.

Parameter	Type	Default Value	Description
Basic			
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Default Maximum Delay	Integer	5 Seconds	The default maximum number of milliseconds to delay malicious requests.
Default Minimum Delay	Integer	500 MilliSeconds	The default minimum number of milliseconds to delay malicious requests.

6.8. Warning Processor

The warning processor is designed to allow a warning message to be presented to a user without completely blocking site access. The warning processor only enables the ability to respond to a user with a "warning", which would allow them to continue browsing the page and the site. The warning would be created and activated for a user by the auto response system, or manually from the console. The existing processor overlays semi-transparent HTML elements on top of the entire webpage, which temporarily disables any mouse or keystrokes on the page and, therefore, creating a "modal dialog" effect. This processor isn't designed to completely stop an attacker from using the website; it is there to warn them. Given the browser debugging tools available today, an attacker may be able to dismiss the warning by means of such tools. Any tampering with the warning's default dismissal behavior (waiting 5 seconds until dismissal button is automatically enabled and clicking on dismiss button) will be considered an incident and will be tracked.

Configuration

Table 6.25.

Parameter	Type	Default Value	Description
Basic			

Parameter	Type	Default Value	Description
Processor Enabled	Boolean	True	Whether traffic should be passed through this processor.
Advanced			
Incident: Warning Code Tampering	Boolean	True	The user has attempted to dismiss the warning without waiting the delay and using the provided mechanism. This is probably an attack on the warning system.
Default Warning Message	String	"Your connection has been detected performing suspicious activity. Your traffic is now being monitored.	The default message to use in the warning dialog. This can be defined on a session by session basis, but if no explicit value is assigned to the warning, this value will be used.
Default Warning Title	String	Security Warning	The default title to use in the warning dialog. This can be defined on a session by session basis, but if no explicit value is assigned to the warning, this value will be used.
Dismissal Delay	Integer	10 Seconds	The amount of time in seconds that must elapse before the warning can be dismissed. This is a soft limit, as an experienced user may be able to get around enforcement measures.
Dismissal Resource	Configurable	Random	The information needed to define the URL and response used to dismiss a warning.
Warning Directory	String	Random	The name of the directory where the warning Javascript and css code will be served from. For example: warningcode.

Incidents

Incident Name: Warning Code Tampering
Complexity: Extreme
Default Response: 1x = Warn User and Logout User, 2x = 5 Day Block.

Cause: MWS is capable of issuing non blocking warning messages to potentially malicious users. These warning messages are designed to force the user to wait for a period of time, before they can dismiss the warning and continue using the site. If the user attempts to exploit or bypass this delay mechanism in order to dismiss the warning early, this incident will be triggered.

Behavior: Once a hacker has been warned, they are then aware that a security appliance is monitoring their activity. This may cause some hackers to investigate what might be protecting the site. This could involve additional scanning, or it could involve attacking the warning mechanism directly. This type of behavior generally indicates a hacker with moderate to advanced skill levels. Depending on what they modify the warning code input to be, this could represent a simple exploratory test, or the user could be trying to launch a more complex attack against he warning code handler itself, such as "Buffer Overflow [http://projects.webappsec.org/Buffer-Overflow]", "XSS [http://projects.webappsec.org/Cross-Site+Scripting]", "Denial of Service [http://projects.webappsec.org/Denial-of-Service]", "Fingerprinting [http://projects.webappsec.org/Fingerprinting]", "Format String [http://projects.webappsec.org/Format-String]", "HTTP Response Splitting [http://projects.webappsec.org/HTTP-Response-Splitting]", "Integer Overflow [http://projects.webappsec.org/Integer-Overflows]", and "SQL injection [http://projects.webappsec.org/SQL-Injection]" among many others.

Chapter 7. Reporting

Mykonos Web Security has a built-in reporting interface that can be accessed through the configuration UI, by clicking on the “Reports” button. Administrators can select from a list of predefined reports that can be run and exported to either PDF or CSV formats.

1. Generate Report

This option generates a report of Mykonos Web Security data from one of the predefined templates. Available templates include:

- Incidents by Type for IP
- Scorecard
- Incident List
- Incidents with Requests and Responses by IP
- Incidents with Requests and Responses by Profile
- Incidents by Type

1.1. Report Schedules

Allows administrators to view all of the reports currently scheduled to run on the system, edit an existing report schedule, or enable or disable a report scheduled to run.

1.2. Scheduling a Report

Administrators can configure the reporting interface to generate a custom report on a schedule. These can be automatically emailed to any email address as specified. To create a scheduled report, select the “Reports” tab following by the "Reports Scheduled" tab. Then click the “Add New Schedule” button at the top of the reporting UI. Enter the following details and then select “OK” and save the change:

Schedule Name	The name of the report schedule that will appear in the reporting interface.
Report Name	The name that will be given to the generated report.
Run	The time schedule in hours, weeks, months, or years that the report should run on.
Period	The period of time that the report should be run on.
format	The format that will be used to generate the report. Options include: PDF, CSV, HTML.
Send to	The email address that this report should be sent to.
Enabled	Sets this report schedule to active or inactive.

1.3. Report History

An archive option, which allows administrators to view all of the historical reports that have been run on the system.

Chapter 8. Using the Security Monitor

The Security Monitor is the interface used to display the current activity, incidents and sessions being monitored by MWS. It can be accessed through a web browser by navigating to "https://THE-MWS-IP:8080". The first screen of the console is the dashboard, which contains the navigation menu and the incidents, sessions and profile graphs. Clicking on any of these graphs or graphical itema in the navigation menu will bring up a corresponding widget which can be used to drill down to get more information.

1. Navigation Menu

The Navigation menu, found on the left hand side of the security monitor, displays a clickable list of any of the currently open widgets, allowing users to quickly switch between the different menus and widgets. The number of menus that appear on the dashboard can be configured by clicking on the "items" dropdown at the top right of the navigation menu, up to four menus can be placed on the dashboard at once.

2. Dashboard

The Dashboard provides a quick overview of the current detected and most frequent attacks, active and top hackers, and the counter responses that have been deployed. Clicking the profiles listed under the "Top Hackers" list will bring up the details of that profile.

3. Incidents, Sessions, and Profiles Charts

These graphs provide an overview of the number of incidents, sessions, and profiles that MWS has logged. By default the graphs will display the totals for each section. Hovering the mouse over any point where the graph intercepts a day will show the count for that day. Clicking on any of the graphs will launch the corresponding widget.

4. Incidents Menu Filter Options

Graph	Configuration	Description
Incidents	Severity	Only display results for the last: 12hrs, 24hours, 7 days, 14 days, or 32 days.
	Count	Displays incident count for the last: 12hrs, 24hours, 7 days, 14 days, or 32 days.
	Severity(pie)	Display results in a pie chart for the last: 12hrs, 24hours, 7 days, 14 days, or 32 days.
Sessions	Count	Displays the sessions created for the last: 12hrs, 24hours, 7 days, 14 days, or 32 days.
Profiles	Count	Displays the profiles created for the last: 12hrs, 24hours, 7 days, 14 days, or 32 days.

5. Complexity Definitions

Complexity is a rating of the skill, effort, and experience necessary to trigger a specific incident. The following is a description of the rating system:

Complexity	Description
Low (Suspicious)	These incidents can be triggered without any special tools or hacking techniques. It is possible, though unlikely, that a non-malicious user may accidentally exhibit the behavior necessary to trigger the incident. For example, altering the contents of the query string could be the results of a sloppy copy/paste of a URL.
Medium (Passively Malicious)	The incident can be triggered only with the use of tools or less commonly used browser features. There is no chance that a legitimate user will accidentally trigger the incident. Incidents of this complexity are primarily triggered by pre-fabricated scripts and scanners, and therefore do not reflect the experience of the hacker. It would be easy for a user with malicious intent to trigger an incident of this complexity without actually understanding anything they are doing. For example, making a request with the OPTIONS http method, which is usually a step performed by most vulnerability scanners.
High (Directly Malicious)	These incidents can be triggered only with the use of tools or advanced browser features. There is no chance that a legitimate user will accidentally trigger the incident. Incidents of this complexity generally require a deep understanding of web vulnerabilities in combination with independent specialized tools. It would be fairly difficult for a malicious user to trigger incidents of this complexity without truly understanding the nature of the potential vulnerability. For example, calling a JavaScript AJAX method manually from firebug.
Extreme (Targeted)	These incidents can be triggered only with the use of tools and advanced browser features. There is no chance that a legitimate user will accidentally trigger the incident. Incidents of this complexity require a very deep understanding of web vulnerabilities, security tools, and applied hacking techniques. These incidents are extremely difficult for an inexperienced user to trigger. For example, using a tool such as John-The-Ripper to crack the encryption used on an .htpasswd file.

6. Incidents List Widget

The Incidents list provides a list of the all the Incidents currently being tracked on the system. Selecting any incident from the Incidents list will launch the incidents widget, which provides a detailed view of the incident. The Incidents list provides the following information:

6.1. Incidents List Details

Table 8.1.

Column	Description
Incident Name	The name of the incident that was triggered.
Profile	The hacker profile associated with this incident.
Severity	The severity level associated with this form of attack.
Complexity	The relative difficulty of this type of attack.

Column	Description
Time	The time at which the incident took place.
Pagination	If there are multiple pages worth of incident data, clicking here will either bring up the next page in the incidents list or the previous page. Users can also click the page number; enter a new value in the text field and hit the enter key, which will skip to the desired page.
Refresh Timer	Sets the refresh time on the incidents list or can be used to manually refresh the list.

7. Incident Widget

The Incident widget provides users with a detailed account of each incident that is being tracked. Users can view the content of the request that triggered the incident, the response data, and customize the assigned severity level. The Incident widget provides the following information:

7.1. Incidents Widget Details

Table 8.2.

Item	Description
Description Tab	Provides a description of the incident.
Details	Allows users to view the details associated with this incident.
Request	Displays all of the details of the request header for this incident.
Response	Displays the header details of the response for this incident.
View Content Body	This option is available from the "Request" and "Response" tabs and allows users to view the content of this request or the content of the response returned to the client.
Name	Name that the system has assigned to the hacker profile.
Threat	The level of threat that this hacker poses. This rank is based on the complexity of the hacking attempts that associated with this profile.
Location	The location that the request came from.
User Agent	The browser and OS that the computer submitting the request is using.

Note

Clicking on any of the orange text in the incident details will launch the appropriate widget and allow the user to drill down into an event for more detailed information. Clicking the orange configuration icon will allow the user to change the complexity level that is assigned to this incident.

8. Sessions Widget

The session widget stores the data that is logged during different hacker sessions.

The information that is logged includes:

- IP Address
- Location
- Profile name
- Time since last activity
- The number of requests
- Errors

Selecting the Incidents, Environments, or Locations tab will load the low level data for the selected session from those widgets.

9. Profile List Widget

The Profiles list widget provides a list of the all the profiles that have been created and that are currently active in the system. Selecting any profile from the list will launch the profile widget, which provides a detailed view of the each user profile. The list provides the following information:

9.1. Profiles List Details

Table 8.3.

Column	Description
Name	The name of the profile.
Description	A customizable description of this profile.
Response	A list of the auto responses that have been deployed on this profile.
Threat	The threat level posed by this profile.
First Active	The first time this profile created an incident on the system.
Last Active	The time that this account was last active on the system

10. Profile Widget

The profile widget stores the data that is logged for each profile on the system. Administrators can use this widget to see details on the hacker such as locations, sessions, responses, and can create a custom description of a profile with any extra notes and information.

Users can drill down into a profile through any of the tabs on the profile widget for greater detail. The default information that is logged includes:

Table 8.4.

Item	Description
Name	The name associated with this profile.
Threat	The level of threat this hacker poses to the system.
Last Active	The last time the hacker was active on the system.
First Active	The first time this profile was active on the system.
Complexity	The complexity of the attack this profile used.
Count	The number of times this account has been active on the system.
First Time	The first time that the incident took place.
Last Time	The last time that the incident took place.

The profiles widget gives users a view into all of data that the appliance has gathered on a hacker. The profile widget divides profile information across 6 tabs:

Incidents Tab — The incidents tab gives a quick breakdown of the incidents and their details that are associated with this profile.

Sessions Tab — This tab provides a quick break down of the sessions and their details that are associated with this profile.

Locations Tab — This tab will show the remote address, host name, city region, postal code, country and the number of requests that are associated with this profile.

Environments Tab — Gives a list of the environments details, such as, browser name, version, OS and the number of requests from this profile. Double clicking anything in this list will launch the environments widget.

Responses Tab — Displays a list of the default auto responses that are active for this profile. This menu also provides a means for deactivating or changing an auto response.

Descriptions Tab — Shows a description for this profile, if an administrator has created any.

10.1. Editing a Profile

Next to the profile picture you will see a small gear shaped icon. Clicking on the icon brings up the menu and will allow administrators to edit the profile name, description associated with this profile, and the avatar.

10.2. Configuring Responses

By clicking on the "Responses" tab next in the profile widget, users will be given the option to configure how the appliance responds to a hacker and view the details of any currently active responses.

Table 8.5.

Option	Description
Warn User	The suspected hacker will be sent a pop-up with a warning letting them know MWS is tracking them.

Option	Description
Block User	The suspected hacker’s access to the site will be removed and they will be blocked.
Force Captcha Validation	The suspected hacker will be presented with a captcha the must be answered to continue using a site.
Inject Header	The suspected hacker's requests will have a custom header injected into them.
Slow Connection	The suspected hacker's request will be delayed by a configurable value.
Strip Inputs	The query parameters and form fields will be stripped from requests issued by the suspected hacker.
Logout User	Terminates any current user sessions for this profile on a site.

Note

After selecting a response, there will be no "on screen notification" for what option was selected until the console screen refreshes. Even though the notification for selecting a response is not immediately displayed, the response itself is immediately active.

11. Location Widget

The locations widget records the IP address, location, the number of requests, and the last time that a hacker profile was active. Locations are displayed on a global map and administrators have the option to zoom in on the location of a hacker's IP address.

- Profile
- Incident
- Session
- Environment

Double clicking on any of the records listed on these tabs will launch the corresponding widget allowing a drill down for more information.

12. Environment Details

The environment widget provides details on a hacker’s user environment. The security console will gather user information on any hacker that creates an incident and keep a list of them here. The information gathered includes:

- Browser
- Browser version
- Operating system

- Number of requests made
- Time of last request

Chapter 9. Auto-Response Configuration

An auto response is composed of a set of rules which define the conditions under which a counter response should be automatically created and activated for a specific session or profile. It is possible to have as many rules as needed to protect the system. However, the more rules, the longer it will take to determine if a new incident matches an event condition. In addition, the more conditions in the rule, the longer the rule will take to evaluate if the event condition matches a new incident.

Table 9.1.

Default Auto Response	Description
Session Management	This auto response rule triggers if the user attempts to manipulate the mykonos session tracking cookie. A first attempt will result in a forced logout of the application, and subsequent attempts will result in progressively longer blocks.
User Agent Processor	This auto response rule triggers when the user attempts to use the same session information between multiple browsers. After this happens 2 more times, the user will first be required to complete a captcha. Then if they do it again, they will be warned, and their connection will be progressively slowed.
Access Policy Processor	This auto response rule triggers if the user attempts to exploit the fake service exposed by this processor. The user will be automatically blocked for 5 days.
ETag Beacon Processor	This auto response rule triggers if a user attempts to manipulate the mykonos cached based tracking token. The user's connection will be progressively slowed down.
Basic Authentication Processor	This auto response rule triggers when the user attempts to exploit the fake .htaccess file exposed by this processor. Simple exploration will result in the user's connection being slowed. Brute force attempts will be stopped with a captcha or block, and successful breaches of the fake protected resource will result in a permanent block.
Robots Processor	This auto response triggers when the user or malicious spider uses the information in the robots.txt file for illegitimate purposes. The first incident results in a captcha, and subsequent violations result in a 1 day block.
Hidden Input Form Processor	This auto response rule triggers when the user modifies a hidden form input parameter. The first attempt will result in the users connection being slowed down. The second attempt will result in a forced logout of the web application, and a

Default Auto Response	Description
	third attempt will cause all of the users input to be stripped for all future requests to the web application. Attempts to change the content type of the form will result in a permanent block.
Cookie Processor	This auto response rule triggers when the user attempts to manipulate the value of a cookie. The first attempt will result in the connection to the website being slowed down. The second attempt will result in a block.
AJAX Processor	This auto response rule triggers when the user interacts with a fake AJAX function injected into the web application. If the user reverse engineers the code and manually invokes its behavior, such as would happen with an automated script or spider, the first incident will result in a forced captcha and a slower connection, and the second attempt will result in a 10 minute block. If the user actually invokes the Javascript function, the first invocation will result in a slower connection, while a second invocation will result in a 10 minute block.
Header Processor	This auto response rule triggers when the user has unusual headers or header data which a normal browser or well developed spider would not supply. If the user excludes required headers such as Host and UserAgent, their connection will be slowed and they will be required to complete a captcha. If the user manipulates their user agent header, they will first be forced to complete a captcha, on a second offense, they will be warned and have their connection slowed. Attempts to overflow headers beyond RFC standards will cause a 1 day block.
Hidden Link Processor	This auto response rule triggers when a spider or malicious user attempts to identify unreferenced resources in a fake directory. Any attempt to perform a directory index or fish for valid file names will cause the connection to be slowed and the user to be blocked for 1 day.
Query Parameter Processor	This auto response rule triggers when a user manipulates the fake query parameter injected by MWS more than 3 times. The users connection will be slowed and they will be automatically blocked for 1 day.
Method Processor	This auto response rule triggers when a user or spider sends a request with a malicious HTTP method such as TRACE. Violations will cause the users connection to be slowed and they will be blocked after 10 minutes for 1 day.
Error Processor	This auto response rule triggers when a user attempts to find unreferenced resources by guessing

Default Auto Response	Description
	file names. Repeat attempts to find nonexistent resources will result in a 1 day block.
File Processor	This auto response rule triggers when a user attempts to find sensitive files by guessing file names or changing parts of valid file names. After enough attempts, the user will be blocked for 5 days.
Warning Processor	This auto response rule triggers when a user attempts to automate the dismissal of the warning response. Attempts to do so will result in a 5 day block.
Cookie Protection Processor	This auto response rule triggers when a user attempts to modify the web application session cookie. The first attempt will result in a warning and a forced logout of the web application, a second attempt will result in a 5 day block.
Captcha Processor	This auto response rule triggers when a user attempts to find a way to bypass the captcha response without solving the captcha. Most attempts will result in a warning, followed by a block on repeat attempts. More serious attempts, such as attempting a buffer overflow on the captcha code will result in an immediate block. In some cases the users connection will also be slowed.
CSRF Processor	This auto response rule triggers if a user attempts to manipulate the CSRF protection introduced by MWS, potentially to find a filter evasion vulnerability. The first attempt will result in a forced captcha, while subsequent attempts will block the user for 1 day.
Custom Authentication Processor	This auto response rule triggers if a user attempts to exploit the authentication mechanism offered by MWS. The first attempt will result in a warning, and subsequent attempts will result in either a direct block for 1 day, or a captcha and then a block. Brute force attempts will first result in a captcha, and subsequent violations will result in a 1 day block.
Client Beacon Processor	This auto response rule triggers when the user attempts to tamper with the client side tracking logic. The user will automatically be blocked for 1 day after a delay of 10 minutes.
New and Modified Profiles	This auto response rule sends out an alert any time a new profile is created, or a profile elevates its threat level. The severity of the alert will equal the threat of the new or elevated profile that triggered the alert.
Returning Profile	This auto response rule sends out an alert any time a profile returns on a subsequent day. For example, a new hacker is observed on Monday, if the hacker is only active for 1 hour on Monday, but returns on Tuesday to continue, this rule will issue an alert. The

Default Auto Response	Description
	severity of the alert will equal the threat level of the profile.
New Incident	This auto response rule sends out an alert any time a new incident is observed. The severity of the alert will equal the complexity of the incident.
New Response	This auto response rule sends out an alert any time a new counter response is activated. The severity of the alert will always equal 1.

1. Using the Editor

To create an auto response, open the configuration UI and select the "ADD New Rule" button. This will launch the editor which can be used to create and edit an auto response:

Table 9.2.

Field	Description
Name	The name of the auto response.
Description	Description of the auto response and its triggers.
Enabled	Sets an auto response to be active.
Safe Mode	allows the auto response to activate, but does not actually respond. This setting is for testing and debugging auto responses.
Code	The actual code that defines the auto response.
Events	the events that will trigger the auto response.
Log	Displays all of the current log information for an auto response.
API Reference	A link to the Auto response API documentation.

Chapter 10. Command Line Interface

The Mykonos Security Appliance contains a Command Line Interface (CLI), which provides users with access to the primary data sources. The CLI allows users to generate complex reports based on the data gathered by the security engine for the purpose of reporting and data analysis. The interface can be run by executing the following command:

```
sudo mykonos-reports-cli
```

To generate a report, users need to specify the following:

- A data source.
- An optional format.
- Any parameters necessary to filter the data.
- Any other output arguments.

When fully constructed, a command to generate a report would look similar to this:

```
sudo mykonos-reports-cli -d=Datasource --format=true/false --parameter
```

Using the –h option in the CLI will bring up a help page with all of the available filters and parameters; a list of parameters is also available at the end of this document.

1. Supported arguments

The command line interface provides access to the raw information that MWS uses. The following are the main arguments provided for accessing, outputting, and formatting the data returned by the CLI:

-d=<data source name>	Define which data source to get data from (from the list of available data sources).
-h	Get help documentation for the specified data source. This includes the accepted arguments and resulting fields.
-o=<path>	Output data to a specified file (if not specified, then output is to console).
-f=<text|csv|xml|html>	The format to use when outputting the data from the data source, supported formats are: CSV, XML, HTML or text.
-l	Get the list of all supported data sources and their names.
-i=<index>	Specify the starting index for the result set.
-m=<max>	Specify the maximum number of records to return.
-s=<fields>	Specify the sorting to apply to any data being read from the data source.

1.1. Data Sources

The CLI provides users with the following data sources In order to access the data generated by the security engine and generate reports. The full list of Data sources can be viewed at any time with the -l option and a Data source can be loaded through the -d= argument:

Browser	Data source that exposes information about known and detectable browsers.
Country	Data source that exposes information about known countries.
Environment	Data source that exposes information about environments.
Incident	Data source that exposes information about incidents.
IncidentType	Data source that exposes information about known incident types.
Location	Data source that exposes information about locations.
OperatingSystem	Data source that exposes information about known and detectable operating systems.
Profile	Data source that exposes information about profiles.
Session	Data source that exposes information about sessions.

1.2. Formatting

All reports are generated with the default columns and sorting. Users may reformat any report or change the sorting for the columns by supplying any optional filtering arguments when the report is generated. To generate a report that has all of the profile and session count data from the session's data source, users would include the following arguments in the command to add those columns:

```
sudo mykonos-reports-cli -d=Session --include-default=true --include-counts=true
```

To generate a report that excludes the default columns, but still contains the session counts data, users would simply run this command with the default option set to false:

```
sudo mykonos-reports-cli -d=Session --include-default=false --include-counts=true
```

1.3. Example Report

The CLI allows users to combine filtering arguments for data sources to create complex and detailed reports. These arguments follow the format:
`--argumentName="Value"`

For example, if a user wanted a report that displayed a list of all the sessions created after 09/12/2010, that also were from a client that used Windows XP, the following algorithm would be used:

```
sudo mykonos-reports-cli -d=Session --createdStartDate='09/12/2010 01:01 am' --environmentId=200
```

Appendix A. Incident Methods

Note

Parameters included in [ ] are optional.

Table A.1.

Name	Description	Parameters
isIncidentType	Check the incident type by either its code or its name.	incident:string
isIncidentDate	Check to see if an incident occurred on the given month, day and year. The month, day and year arguments may be left empty to match any value.Note that Jan = 1, and years are in the format YYYY.	[month:int] [day:int] [year:int]
isIncidentDateRange	Check to see if an incident occurred between two dates. All values must be defined. Note that Jan = 1, and years are in the format YYYY.	start_month:int start_day:int start_year:int end_month:int end_day:int end_year:int
isIncidentTime	Check to see if an incident occurred at a given time. The hour, minute and second arguments may be left empty to match any value.	[hour:int] [minute:int] [second:int]
isIncidentTimeRange	Check to see if an incident occurred between a given time range. All values must be specified.	start_hour:int start_minute:int start_second:int end_hour:int end_minute:int end_second:int
isIncidentCount	Check the number of times an incident has occurred against an integer operation and specified value. Supported operations include (>, <, ==, !=). The results are: (count [operator] value)	operator:string value:int

Name	Description	Parameters
isIncidentCountRange	Check to see if the number of times an incident has occurred is within a given range.	min:int max:int
isIncidentContextSubString	Check to see if the context XML associated with the incident contains the provided substring. The search is case sensitive by default, unless the second parameter is "false"	search:string [[caseSensitive]:Boolean]
isIncidentContextPattern	Check to see if the context XML associated with an incident contains a simple pattern. Supported pattern wild cards include +, ? and *. Pattern matches are performed case sensitive unless the second parameter to this method is "false".	pattern:string [[caseSensitive]:Boolean]
isIncidentIP	Check to see if an incident came from a given IP address. Each parameter specifies the required value for the specific block of the address. Any of the parameters can be left empty to match any value.	[a_block:int] [b_block:int] [c_block:int] [d_block:int]
isIncidentIPRange	Check to see if an incident came from a given IP address range. Each parameter specifies a range of accepted values for the specific address block. Ranges are specified in the format: min-max. For example: 10-22, or 0-255	[a_block_range:string] [b_block_range:string] [c_block_range:string] [d_block_range:string]
isIncidentBrowser	Check to see if the incident occurred from a given browser. The parameter expects the canonical name of the browser.	name:string
isIncidentOperatingSystem	Check to see if the incident occurred from a given operation system. The parameter expects the canonical name of the operating system.	name:string
isInicdentBrowserVersion	Check to see if the incident occurred from a specified version of the browser. The check is case sensitive by default, unless the second parameter is "false". The version could contain any character and should be considered as an arbitrary user supplied string value.	version:string [[caseSensitive]:Boolean]

Name	Description	Parameters
isIncidentBrowserVersionPattern	Check to see if the incident occurred from a browser with a version that matches a given simple pattern. Pattern wild cards >include ?, * and +. The match is done case sensitive unless the second parameter is "false". The version could contain any character and should be considered as an arbitrary user supplied string value.	pattern:string [[caseSensitive]:Boolean]
isIncidentBrowserVersionSubString	Check to see if the incident occurred from a browser with a version that contains the given sub string. The match is done case sensitive unless the second parameter is "false". The version could contain any character and should be considered as an arbitrary user supplied string value.	Search:string [[caseSensitive]:Boolean]
isIncidentCountry	Check to see if the incident originated from a given country. The parameter expects a valid 2 character country code, or the canonical name of the country.	country:string
isIncidentLatitude	Check to see if the incident originated from a specified geographical latitude. The parameter is expected to be a decimal number between -90.0 and +90.0	latitude:float
isIncidentLatitudeRange	Check to see if the incident originated between a specified geographical latitude range.The parameters are expected to be decimal numbers between -90.0 and +90.0.	min:float max:float
isIncidentLongitude	Check to see if the incident originated from a specified geographical longitude. The parameter is expected to be a decimal number between -90.0 and +90.0	longitude:float
isIncidentLongitudeRange	Check to see if the incident originated between a specified geographical longitude. The parameters are expected to be decimal numbers between -90.0 and +90.0.	min:float max:float

Name	Description	Parameters
isIncidentCity	Check to see if the incident originated in a specified city. The parameter is expected to be the city name and is case sensitive unless the second parameter is "false"	city:string [caseSensitive]:Boolean
isIncidentCityPattern	Check to see if the incident originated from a city that matches a specified pattern. The supported wild cards are *>, ?, and +. The pattern is case sensitive unless the second parameter is "false".	pattern:string [caseSensitive]:Boolean
isIncidentCitySubString	Check to see if the incident originated from a city that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false"	search:string [caseSensitive]:Boolean
isIncidentHost	Check to see if the incident originated in a specified host. The parameter is expected to be the host name and is case sensitive unless the second parameter is "false"	host:string [caseSensitive]:Boolean
isIncidentHostPattern	Check to see if the incident originated from a host name that matches a specified pattern. The supported wild cards are *, ?, and +. The pattern is case sensitive unless the second parameter is "false".	pattern:string [caseSensitive]:Boolean
isIncidentHostSubString	Check to see if the incident originated from a host name that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false"	search:string [caseSensitive]:Boolean
isIncidentRegion	Check to see if the incident originated in a specified region. The parameter is expected to be the region name and is case sensitive unless the second parameter is "false"	region:string [caseSensitive]:Boolean
isIncidentRegionPattern	Check to see if the incident originated from a region that matches a specified pattern. The supported wild cards are *, ?, and +. The pattern is case sensitive	pattern:string [caseSensitive]:Boolean

Name	Description	Parameters
	unless the second parameter is "false".	
isIncidentRegionSubString	Check to see if the incident originated from a region that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false"	search:string [caseSensitive]:Boolean
isIncidentZip	Check to see if the incident originated in a specified zip code. The parameter is expected to be the zip code and is case sensitive unless the second parameter is "false". While zip codes should generally be numeric, there is the possibility of foreign zip codes containing strange characters.	zip:string [caseSensitive]:Boolean
isIncidentZipPattern	Check to see if the incident originated from a zip code that matches a specified pattern. The supported wild cards are *>, ?, and +. The pattern is case sensitive unless the second parameter is "false".	pattern:string [caseSensitive]:Boolean
isIncidentZipSubString	Check to see if the incident originated from a zip code that contains a specified sub string. The substring search is done case sensitive unless the second parameter is "false"	search:string [caseSensitive]:Boolean